



Rocket Model 204 Janus/TN3270 Debugger

User's Guide

Version 7.5

November 2014
JDB-0705-UG-62

Notices

Edition

Publication date: November 2014

Book number: JDB-0705-UG-62

Product version: Version 7.5

Copyright

© Rocket Software, Inc. or its affiliates 2006-2014. All Rights Reserved.

Trademarks

Rocket is a registered trademark of Rocket Software, Inc. For a list of Rocket registered trademarks go to: www.rocketsoftware.com/about/legal. All other products or services mentioned in this document may be covered by the trademarks, service marks, or product names of their respective owners.

Examples

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

License agreement

This software and the associated documentation are proprietary and confidential to Rocket Software, Inc. or its affiliates, are furnished under license, and may be used and copied only in accordance with the terms of such license.

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

Corporate Information

Rocket Software, Inc. develops enterprise infrastructure products in four key areas: storage, networks, and compliance; database servers and tools; business information and analytics; and application development, integration, and modernization.

Website: www.rocketsoftware.com

Rocket Global Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA

Contacting Technical Support

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

Email: m204support@rocketsoftware.com

Telephone :

North America +1.800.755.4222

United Kingdom/Europe +44 (0) 20 8867 6153

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

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

www.rocketsoftware.com/support

Contents

Chapter 1: Introduction	1
About the Debuggers	2
Getting started	7
Chapter 2: Getting Acquainted with the Client GUI	9
The tabbed pages	10
The lower windows	14
The menu bar	17
The File menu options	18
The Window menu options	24
The Search menu options	26
The Breakpoints menu options	27
The Execution menu options	29
The Data Display menu options	33
The Error menu options	35
The Macros menu options	36
The Help menu options	38
The button bar	39
The search facility	44
The tracing options	48
The value displaying controls	49
The Status bar	50
The Entity-name input box	51
Chapter 3: Performing Basic GUI Tasks	53
Controlling the execution of program code	54
Step, Step Over, and Run	55
Using breakpoints	57
Suppressing the break at the end of request evaluation	61
Breaking after READ SCREEN or READ MENU statements	62
Stepping out	64
Cancelling execution	65
Excluding sections of source code from debugging	66
Selectively excluding source code blocks	67
Running to a specific procedure	75
Running only to listed procedures	79
Altering the flow of execution	83
Previewing program code	85
Viewing and modifying program elements	87
Watching program data items	87
Adding and removing Watch Window items	88

Saving and restoring Watch Window contents	91
Getting a detailed view of the value of a watched item	93
Watching Model 204 fields	95
Watching global variables	96
Watching object variables	97
Watching \$lists, Stringlists, and Arraylists	98
Watching class member Variables	100
Displaying temporarily the value of a program data item	101
Displaying \$lists, Stringlists, and Arraylists	104
Displaying Janus SOAP XML document objects	106
Displaying all Variables of an object's class	111
Displaying the current occurrence value in an FEO loop	116
Displaying all fields in a record	117
Displaying Model 204 parameters	122
Displaying the Universal Buffer content	123
Setting the value of a variable	124
Getting source file, audit trail, and web buffer information	126
Locating and editing procedure source files	126
Viewing dummy string variables	127
Viewing the audit trail	128
Viewing the web output buffer	129
Tracing program execution	129
Tracing all lines executed	130
Tracing all updates to a variable's value	132
Tracing until a value change or until a value match	133
Displaying a statement history	134
Viewing programs that contain coding errors	138
Debugging requests that spawn daemons	141
Debugging Web Server persistent sessions	144
Debugging multiple Web Servers	147
Chapter 4: Additional Debugger Functionality.....	149
Copying, printing, or saving text	149
Using the SIRIUS DEBUG command	151
Using the SIRIUS DEBUG command for web threads	157
Debugging SSL applications	159
Debugging Web Service applications	160
Using the DebuggerTools class methods	161
Using a local editor	166
Using Xtend with the Debugger	168
Using UltraEdit with the Debugger	172
Chapter 5: The Client Command Reference.....	179
addWatch command	180
addWatchOnCurrentLine command	180
assert command	180
bottom command	183
breakOnNextProc command	184
breaks command	184

breaksAt command	185
buttonBar command	186
cancel command	187
clearAudit command	187
clearBreakpointOnCurrentLine command	188
clearBreaks command	189
clearButton command	189
clearExecutionTrace command	190
clearHistory command	190
clearKey command	191
clearMacroConsole command	191
clearStatus command	192
clearWatch command	192
clearWebBuffer command	193
closeCommandLine command	193
closeExternalAuditTrailWindow command	193
closeExternalButtonWindow command	194
closeExternalExecutionTraceWindow command	194
closeExternalWatchWindow command	194
closeExternalWebBufferWindow command	195
closeExternalWindows command	195
closeHistory command	195
closeMacroConsole command	196
closeValueDisplay command	196
continuelf command	197
continueMacroIf command	198
copy command	200
createMacro command	200
debugPreview command	201
decrement command	201
disableButton command	202
echo command	203
editMacroFromUISelection command	204
enableButton command	204
evaluate command	205
expandList command	205
expandObject command	206
extraButtonBar command	207
feoDisplay command	208
firstHistory command	208
focusToSearchBox command	209
generatePac command	209
getHistory command	211
getVariablesForClass command	211
help command	212
hideLower command	213
include command	214
includelf command	214
increment command	215
jumpToLine command	216

jumpToMatch command	217
kill command	218
labelButton command	219
lastHistory command	220
loadWatch command	220
macro command	221
macroConsole command	221
macroTrace command	222
macroWait command	223
mainButtonBar command	223
manual command	224
mapButton command	224
mapKey command	226
moveBrowserToTop command	227
moveTn3270ToTop command	227
nextCompileError command	228
nextHistory command	228
noSpan command	229
nsLookup command	230
openCommandLine command	231
openExternaAuditTrailWindow command	231
openExternalButtonWindow command	232
openExternalExecutionTraceWindow command	232
openExternalWatchWindow command	233
openExternalWebBufferWindow command	233
openMacroConsole command	234
pafigi command	234
pai command	235
pin command	236
preferences command	236
previousCompileError command	237
previousHistory command	237
reloadBlackList command	238
reloadWhiteList command	238
reloadLists command	239
removeCurrentWatch command	240
resetAssertCounts command	241
resetGlobalAssertCounts command	241
restart command	242
restartDefault command	242
restoreLower command	243
restoreTitle command	243
run command	244
runMacroFromUISelection command	244
runUntil command	245
runUntilVariableChanges command	245
runWithoutDaemons command	246
saveWatch command	246
searchDown command	247
searchFromBottom command	248

searchFromTop command	250
searchUp command	251
selectAuditTab command	252
selectExecutionTraceTab command	253
selectNextTab command	253
selectProcSelectionTab command	253
selectSourceTab command	254
selectWatchWindow command	254
selectWebBufferTab command	254
set command	255
setBlackList command	256
setBreakpointOnCurrentLine command	257
setIEmode command	258
setM204Data command	260
setPreference command	260
setStatusMessage command	262
setTitle command	264
setWhiteList command	265
showAbout command	266
showCommands command	266
showFunctions command	267
showIE command	267
showShortcuts command	268
skipPreview command	268
span command	269
step command	269
stepOut command	270
stepOver command	270
toggle command	271
toggleBreakpointOnCurrentLine command	272
toggleInItExclude command	273
toggleLower command	273
top command	274
trace command	274
traceUntilVariableEqualsValue command	275
traceValues command	275
turnOffBlackList command	276
turnOffDebugging command	276
turnOffWhiteList command	276
turnOnBlackList command	277
turnOnWhiteList command	277
unPin command	278
unSet command	279
valueDisplay command	279
varDump command	280
viewText command	281
windowToTop command	281
Chapter 6: Customizing Client Operations.....	283

Reconfiguring GUI buttons and hot keys	283
Introducing the configurable components	285
Setting up the ui.xml file	287
Default settings of buttons and hot keys	291
Changing the colors in Client displays	293
Changing the location of Client work files	296
Changing the font size in Client displays	299
Opening an external window	300
Hiding the Client's lower windows	305
Seeing through Client windows	307
Chapter 7: Using Debugger Macros.....	309
Creating and running a macro	309
Mapping a macro to a button or hot key	314
Passing a command argument to a macro	314
Using the console and command line	316
Using the Macro Autorun feature	318
Working with macro variables	319
Working with Client functions	321
&&amDaemon function	322
&&arg function	323
&&assertFailureCount function	324
&&assertStatus function	324
&&assertSuccessCount function	325
&&blackOrWhiteList function	326
&&concatenate function	327
&¤tTitle function	330
&&exists function	331
&&getMainSearchInputArea function	332
&&getVariableOrFieldInputArea function	332
&&globalAssertFailureCount function	332
&&globalAssertStatus function	333
&&globalAssertSuccessCount function	333
&&ieMode function	333
&&index function	334
&&isWatched function	334
&&length function	335
&&numberOfBreakpoints function	335
&&numberOfLevels function	336
&&numberWatched function	336
&&originalTitle	337
&&preference function	337
&&procName function	338
&&prompt function	338
&&searchResult function	339
&&searchSuccess function	339
&&selectedTab function	340
&&statusMessage	340

&&substring function	341
&&sum function	341
&&verifyMatch function	342
&&verifyNoMatch function	343
&&>windowStatus function	343
Chapter 8: Problem Diagnosis.....	345
Debugging the Janus Debugger	345
Debugging the Sirius Debugger	348
How the Janus Debugger handles communication breaks	354
How the Sirius Debugger handles communication breaks	357
Tracking Client performance	359
Resolving issues when automatically maintaining IE proxy settings	361
Chapter 9: Installation and Configuration.....	363
Overview	364
Online Configuration	365
Check prerequisites	365
Authorize the Debugger	365
Set Model 204 system parameters	366
Define and start the Debugger Server port	367
Define and start a client socket port (Sirius Debugger only)	368
Workstation Configuration	369
Perform preliminary tasks	369
Run, check, and verify the Client installation	371
Customize the Debugger configuration file	374
Configure the web browser (Janus Debugger only)	380
Test the end-to-end configuration	387
Providing updated versions of the Debugger Client	390
Chapter 10: Release Notes.....	395
Index	427

CHAPTER 1 *Introduction*

This chapter describes how this document is organized, then provides brief product [overview](#)^[2] and [quick-start](#)^[7] subsections.

Note: Most cross-references in this document are underlined (hyperlinked and shown in a blue font if viewed online) and accompanied by a page icon like the following, which frames the number of the page that contains the target of the reference:



Using this document

This document assumes that you have completed the mainframe and workstation tasks described in [Installation and Configuration](#)^[363].

These are the information subdivisions:

- This "Introduction," which includes an overview of the Debugger products and a "getting started" section that outlines how to begin to use the products.
- The [Getting Acquainted with the Client GUI](#)^[9] and [Performing Basic GUI Tasks](#)^[53] chapters are designed to orient you quickly to the Debugger Client and get you working.
- The [Additional Debugger Functionality](#)^[149] chapter describes features or tasks that are not primarily concerned with the Client GUI.
- The [Client Command Reference](#)^[179] chapter provides detailed descriptions of the Debugger Client commands you can use to program the Client interface controls and displays.
- The [Customizing Client Operations](#)^[283] chapter describes how to modify the default arrangement of Client buttons and hot keys, as well as the colors of the text or background of Client displays.
- The [Using Debugger Macros](#)^[309] chapter describes how to define scripts containing one or more of the commands that activate the various Debugger controls.
- The [Problem Diagnosis](#)^[345] chapter provides a closer inspection of how the Debuggers handle errors and get and manipulate the data you see in the Client.
- The [Installation and Configuration](#)^[363] chapter includes product installation and set up information, as well as information about setting up at your site a centralized distribution of updated Debugger Client replacement files.
- The [Release Notes](#)^[395] provide an archive of information about features that are new or enhanced in each build of the Debugger Client.

1.1 About the Debuggers

The *Janus Debugger* is a tool designed for software developers who create and maintain *Janus Web Server* applications. With software installed on the Web Server host *Model 204 Online*, as well as on a workstation with a browser that can access the Web Server, the Debugger lets you examine in statement-by-statement detail the User Language code that the Web Server executes.

The *Sirius Debugger* is designed for developers who create and maintain *Model 204 3270-screen* and *Batch2* applications. With software installed on the host *Model 204 Online*, it uses essentially the same Debugger Client as the *Janus Debugger*, letting you examine 3270-screen and Batch2 code.

An instance of the Debugger Client can be debugging a web application (acting as the *Janus Debugger*), or it can be debugging a 3270/Batch2 application (acting as the *Sirius Debugger*). At any given time, however, it may only be debugging one type of application. It "knows" the type of thread being debugged (the title bar of the Client toggles to reflect the thread type) and it communicates that information to the Online (which ultimately controls the granting of permission to debug).

These subsections continue the overview of the Debuggers:

[The basic operation](#)  2

[The feature set](#)  3

[The architecture](#)  4

[Versions and builds](#)  6

[System requirements](#)  6

[Known limitations](#)  6

The basic operation

When you are running under the *Janus Debugger*, and *Janus Web Server* is about to run a request on your behalf, or you are running under the *Sirius Debugger* and you submit a 3270-screen or Batch2 request:

1. Request source code is sent to the Debugger Client deployed on a workstation.
2. The Debugger Client displays the code, navigable to top and bottom, with search features available.
3. Program execution is paused, the Debugger awaiting the instructions you provide through the Client GUI.

Although the *Janus Debugger* is limited to *Janus Web Server* applications, the programs you debug with the *Janus Debugger* may be, or contain, the following:

- *Janus SOAP* applications
- Sdaemons or transactional sdaemons, including those spawned by other sdaemons or by \$COMMBG requests
- Synchronous \$COMMBG requests, including those spawned by other \$COMMBG requests or by sdaemons
- \$WEB_FORM_DONE persistent mode requests

The *Sirius Debugger* is primarily for debugging User Language programs other than web applications. These include:

- 3270 full screen applications
- Batch2 applications
- HTTP client applications that use the *Janus Sockets* HTTP Helper
- Any other programs run from the *Model 204* command prompt, including those that do line mode input via \$READ.

The feature set

Both Debuggers provide these typical debugging capabilities:

- Viewing code while running it
- Stepping through the code one statement at a time
- Examining variable values
- Using a separate "watch window" for the display of specified data items
- Setting breakpoints at which code execution pauses
- Skipping over selected subroutines

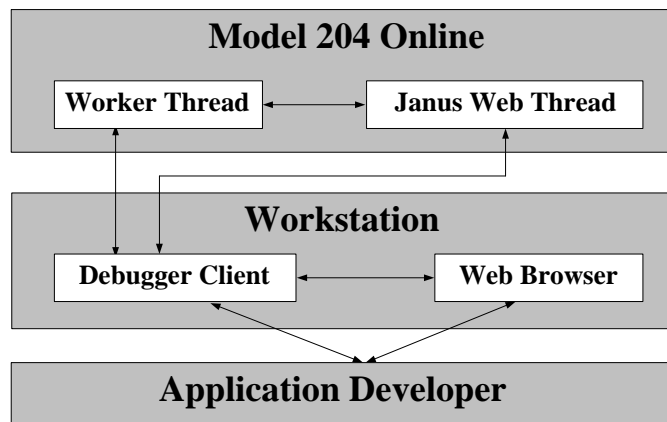
In addition, the Debuggers provide these special features:

- A [compact](#)⁹ Graphical User Interface with user-configurable buttons and hot keys
- Views of the *Model 204* audit trail and code executed by sdaemons
- Tracing of statements that modify variables in executed code lines, or tracing until a variable becomes a certain value
- Saving of sets of variables to be watched, which are restorable at any time
- Display of Web Server output-buffer data (*Janus Debugger*) and screen variables (*Sirius Debugger*)
- A code line's *Model 204* procedure details (subsystem, file, name, line number, text preceding dummy string substitution)

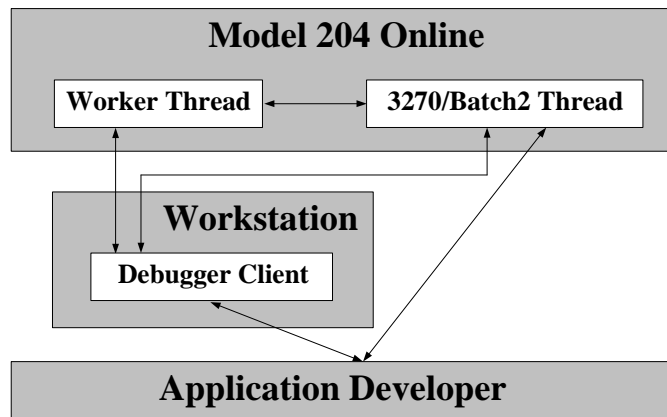
- Filtering of program code to skip over all but the *Model 204* procedure you identify or the procedures you pre-selected (added to "White List")
- Stepping out of of a called subroutine, method, or daemon and continuing processing at the statement after the call to the subroutine, method, or daemon
- Examination of Sirius extensions to User Language (including LongStrings, \$lists, objects)
- System class methods that let you insert code in User Language that runs only under the Debuggers
- Facilitated access for code editors (for example, Xtend®, UltraEdit®)

The architecture

The following figure displays the principal components in a *Janus Debugger* configuration:



The main components in a *Sirius Debugger* configuration parallel those in the *Janus Debugger* configuration, the principal exception being the absence of the web browser:



Per Debugger session for a *Janus Web Server* or 3270/Batch2 program, these components are active in the Online and on the workstation:

Model 204 nucleus hooks

(Assembler; not shown above)

Compiler:

- Capture source lines, before and after dummy string substitution.
- Note source of line (procedure and file).

Sirius Evaluator:

- Can step one statement at a time, and can step over routines and methods.
- Can stop on breakpoints.
- Between statements, can get variable, field, and global values, to support display, trace, watch, etc.

Janus Web thread

Janus Debugger only.

3270/Batch2 thread

Sirius Debugger only.

Worker threads

(Assembler and User Language, one each to the thread being debugged and to the Debugger Client, respectively)

For each debugging session, a “worker thread” arbitrates between the thread being debugged and the workstation-based client.

A worker thread does the following:

- Passes data between the thread being debugged and the Client.
- Controls the execution of the thread being debugged, at the behest of the Client.

Debugger Client

(WIN32 GUI, written in VB.NET)

The Debugger Client resides on the developer's workstation. It is a proxy server in a *Janus Debugger* session: the browser no longer directly connects to the Web Server, but instead it “sends” to and “receives” from the Client.

The Client GUI lets you control the application and view variable values, source code, traces, and for *Janus Debugger* sessions, the web output buffer.

Web browser

Janus Debugger only. Any web browser that allows proxy servers.

Versions and builds

First available in Version 7.0 of the *Sirius Mods*, the Debuggers also include features that are available only with later versions of the *Sirius Mods*. Those features are labeled as such in this documentation and are also mentioned in the Release Notes for the *Sirius Mods*.

Independent of the version of the *Sirius Mods* being run on the *Model 204* host machine, the Debugger Client has its own integer-numbered sequence of "builds" that contain new and updated features. From build to build of the Debugger Client, the **About the Debugger Client** box (accessed by **Help** menu, option **About**) briefly lists the versions of the *Sirius Mods* that support the product and a searchable list of the principal changes in each of the previous builds of the Debugger Client.

System requirements

The [Online Configuration](#)^[365] section specifies the requirements for *Sirius* and *Model 204* product release levels, licenses, and maintenance, as well as Debugger product authorization. In addition are instructions for configuring a *Model 204* Online that hosts the Debugger, which include User 0 parameter and TCP port definitions and increases in storage table and work area sizes for both of these:

- The worker thread that services a debugging session
- The thread on which the program to be debugged is running

Known limitations

The following are known limitations and issues concerning the Janus and *Sirius* Debuggers:

- You may not use either Debugger for **DBCS data**.
- You may use the *Janus Debugger* against a **Janus Web Legacy Support** thread if you also have a license for the *Sirius Debugger*.
- The *Janus Debugger* **Web Buffer** tab only displays **printable output**; binary data is not shown.
- No **asynchronous daemon** or **\$COMMBG requests** may be debugged. They are simply ignored by the Debugger (and they execute normally).
- **Windows 95, Windows 98, and Windows ME** are not supported by the client. You must be running Windows 2000, XP, 2003 Server, Vista, or 7. There are no plans to support Window 95/98/ME.
- On the workstation, you currently must **hand-edit the debuggerConfig.xml file** to configure it after installation or to alter the configuration.
- Both the **Execution Trace** tab (output from one trace) and the **Web Buffer** tab have **5000-line capacities**. If the capacity is exceeded, the last 5000 lines are shown.

1.2 Getting started

In place of a tutorial, it is recommended that you teach yourself to use the Debugger. To get started:

1. Make sure the following are true:
 - The product has successfully been [installed and configured](#)^[363], on the mainframe and on the workstation that is to host the Debugger Client, recently enough that you are confident that the various port numbers and names assigned and, say, your web browser's definition of the Debugger Client as a proxy server (*Janus Debugger*), are still valid.
If you are unsure, you may want to review the installation documentation and/or rerun the [final installation test](#).^[387]
 - You know the Debugger Server server socket port is [started](#)^[367] (and also, only for the *Sirius Debugger*, the Debugger Server client socket port is [started](#)^[368]), and the Debugger Client is [started](#)^[371].
2. Start the Debugger Server worker thread:
 - *Janus Debugger*: This is done automatically by the next step. Go to Step 3.
 - *Sirius Debugger*: From the *Model 204* command line, issue the [SIRIUS DEBUG ON](#)^[151] command. Your session lasts until you log off *Model 204* or issue [SIRIUS DEBUG OFF](#) or its [Debugger Client-command](#)^[285] equivalent ([turnOffDebugging](#)).
3. Run a request you want to debug:
 - *Janus Debugger*: From your web browser, invoke a URL that accesses a program that is run by your *Janus Web Server*.
 - *Sirius Debugger*: From the *Model 204* command line, start a User Language program (Include a procedure or invoke an APSY subsystem).
4. Using the Debugger Client GUI, control the execution of the User Language code.
You might take a few minutes with [Getting Acquainted with the Client GUI](#)^[9], then try some of the operations described in [Performing Basic GUI Tasks](#)^[53].

Accessing and printing Help information

For quick access to Debugger Client Help information (accessed from the **Help** menu or the F1 key), try finding in the Index the name of the GUI control (button, tab, box label) involved in the operation you want to know more about.

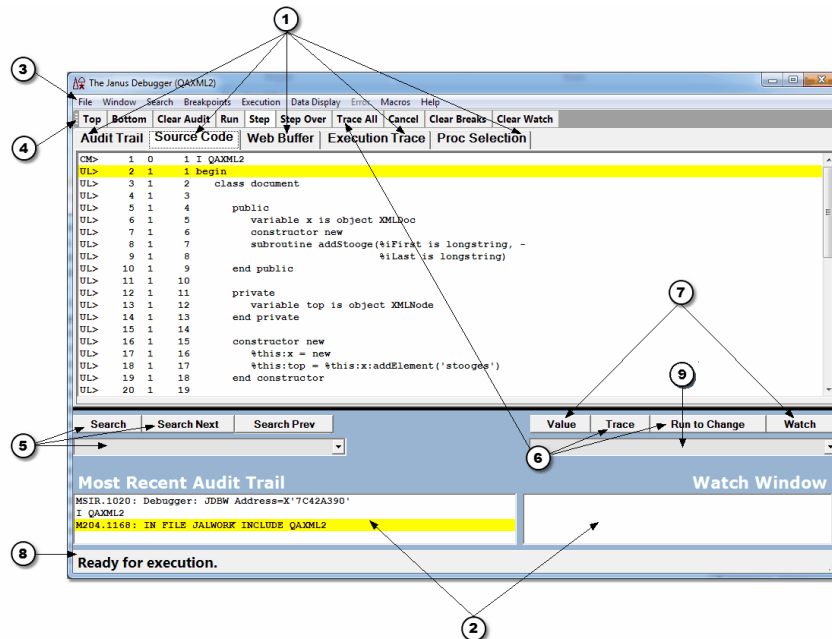
The online Help information is reproduced in PDF format (as this *Janus/Sirius Debugger User's Guide*) in your Debugger Client installation folder. This PDF is immediately accessible by selecting the **View PDF Manual** option from the Debugger Client **Help** menu. It is highly recommended that you use the PDF document as the source for any lengthy Debugger Help printing.

The **Print** option that is available from the toolbar on your Microsoft HTML Help viewer gives reasonable results. To print multiple topics, select **Print** from the toolbar, then select the option to **Print the selected heading and all subtopics**.

CHAPTER 2 *Getting Acquainted with the Client GUI*

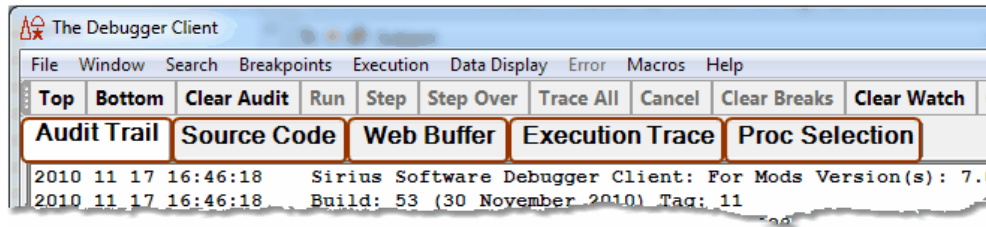
The following image of the Debugger Client (debugging a Janus Web program) has numbered labels that point to the principal areas of the Debugger, which are described in the corresponding numbered sections in the linked list below. The image is also "live": you can simply click any area of the image you want to find out about. In a *Sirius Debugger* session, the **Web Buffer** tab is not present.

1. [The tabbed pages](#)^[10]
2. [The lower windows](#)^[14]
3. [The menu bar](#)^[17]
4. [The control button bar](#)^[39]
5. [The search facility](#)^[44]
6. [The tracing buttons](#)^[48]
7. [The value displaying controls](#)^[49]
8. [The Status bar](#)^[50]
9. [The Entity-name input box](#)^[51]



2.1 The tabbed pages

An important part of the user interface to the *Janus Debugger* or the *Sirius Debugger* is the set of tabbed pages that use the main display window below them (the **Web Buffer** tab is not present for Sirius Debugger sessions; [Daemon tabs](#)^[141] display dynamically if the program invokes daemons):



These **pages are resizable** by mouse if you grab and drag the black border bar at the bottom of the main window.

The **pages are copyable**:

- If you press the Ctrl+C keyboard key combination or select the **Copy** option from the **Window** menu, the contents of the active (topmost) page are copied to the Windows clipboard. The number of lines copied is displayed in the [Status bar](#)^[50]. Only the **Proc Selection** page is not copyable.
- If you [map the viewText command](#)^[285] to a Client button or hot key, or select its equivalent from the **Window** menu, you can [invoke a text viewer](#)^[149] that lets you copy, edit, print, and save text data from Client pages.

Some of the **pages are displayable in windows outside of the Client**. These [external](#)^[300] windows are easily invoked by double-clicking their tabbed page name, by selection from the Client's **Window** menu, or by [mappable command](#)^[283].

Audit Trail tab

The **Audit Trail** page displays, from the beginning of the debugging session:

- The *Model 204* audit trail lines produced by the online thread that is servicing web requests from your browser(s) or by the thread that is servicing your 3270/Batch2 requests
- The *Model 204* audit trail lines produced by any threads that run daemons on behalf of your web or 3270/Batch2 requests
- Information about the state of the Debugger Client, such as the port on which it is listening and the port whose web server requests it is debugging
- All outgoing HTTP messages sent by your browser (passed-through by the *Janus Debugger*)

Audit Trail	Source Code	Web Buffer	Execution Trace	Proc Selection
2006 07 12 21:34:00			Connection From: 127.0.0.1	
2006 07 12 21:34:00			Web request: GET http://global.msads.net/ads/1/replay.swf?fd=dellnet.msn.	
2006 07 12 21:34:00			Web request will NOT be debugged...	
2006 07 12 21:34:00			Response: received 1480	
2006 07 12 21:34:00			Response content-length: 1062	
2006 07 12 21:34:00			Receive of HTTP Response complete, 1480 bytes.	
2006 07 12 21:34:00			Response code: 200 OK	
2006 07 12 21:34:02			Connection From: 127.0.0.1	
2006 07 12 21:34:02			Web request: GET http://sirius-software.com:9219/jalwork/daemon5 HTTP/1.0	
2006 07 12 21:34:02			Web request will be debugged...	
2006 07 12 21:34:03			*****	
2006 07 12 21:34:03			Connected to the Debugging Server: sirius-software.com:3355	
2006 07 12 21:34:03			Model 204 Version: 6.1.0G	
2006 07 12 21:34:03			Sirius Mods Version: 6.9	
2006 07 12 21:34:03			Session name: 034D230656	
2006 07 12 21:34:03			*****	
2006 07 12 21:34:03.15		1	3 LI I DAEMON5	
2006 07 12 21:34:03.15		1	3 MS M204.1168: IN FILE JALWORK INCLUDE DAEMON5	

The [Most Recent Audit Trail window](#)^[14] also *and only* displays audit trail lines.

The Audit Trail page is displayable separate from the Client in an [external](#)^[300] window.

Source Code tab

The **Source Code** page displays the source code lines of the User Language request that is currently being debugged. Here is where most of your interactions with program code take place: setting breakpoints, viewing the current execution position, getting detailed information about a source code line, watching variables from a selected source code line.

Audit Trail	Source Code	Web Buffer	Execution Trace	Proc Selection
CM> 1 0	1 I DAEMON5			
CM> 2 1	1 *nesting new daemon instances and checking their master numbers			
UL> 3 1	2 b			
UL> 4 1	3 Audit 'I am starting'			
UL> 5 1	4 %speed is object daemon			
UL> 6 1	5 %list is object stringList			
UL> 7 1	6 %X is object stringList			
UL> 8 1	7 %speed = new			
UL> 9 1	8 %n is float			
UL> 10 1	9 %n = %speed:usernumber			
UL> 11 1	10 Audit 'n is: ' %n			
UL> 12 1	11 %n = %speed:masternumber			
UL> 13 1	12 Audit 'n is: ' %n			
UL> 14 1	13 %n = %speed:parentnumber			
UL> 15 1	14 Audit 'n is: ' %n			
UL> 16 1	15 %list = new			

Add Watch
Toggle BreakPoint
Procedure Information
FEO OCC IN value
Display %n

The page display format is:

Column 1	Line type: CM> — command UL> — User Language statement ER> — error message BR> — breakpoint
Column 2	Simple line number
Column 3	Include level
Column 4	Line number within include

Web Buffer tab

Not ordinarily present for *Sirius Debugger* sessions, the **Web Buffer** page displays the lines the *Janus Web Server* application is preparing to send to the browser at the completion of the request. The page is updated in real time, each time you break execution, as you step through a code program and PRINT and HTML statements are executed. The most recent lines added to the page are highlighted, and you can watch your output HTML being built.

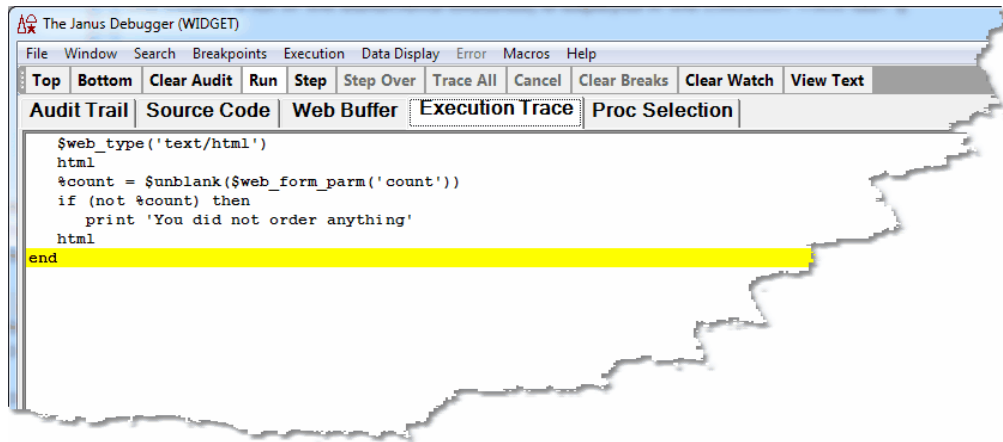
Audit Trail	Source Code	Web Buffer	Execution Trace	Proc Selection
<pre><html> <head> <title>Default Janus Web Home Page</title> </head> <body bgcolor="#ffffd8"> <table cellpadding=2 cellspacing=5 border=0> <tr> <td valign=top haligh=left> </pre>				

Top, Bottom, and search buttons are available. The page is cleared when a new request is initiated or if you clear it manually (via the **Window > Clear Web Buffer** button or the [clearWebBuffer](#)^[193] command).

The **Web Buffer** page is displayable separate from the Client in an [external](#)^[300] window.

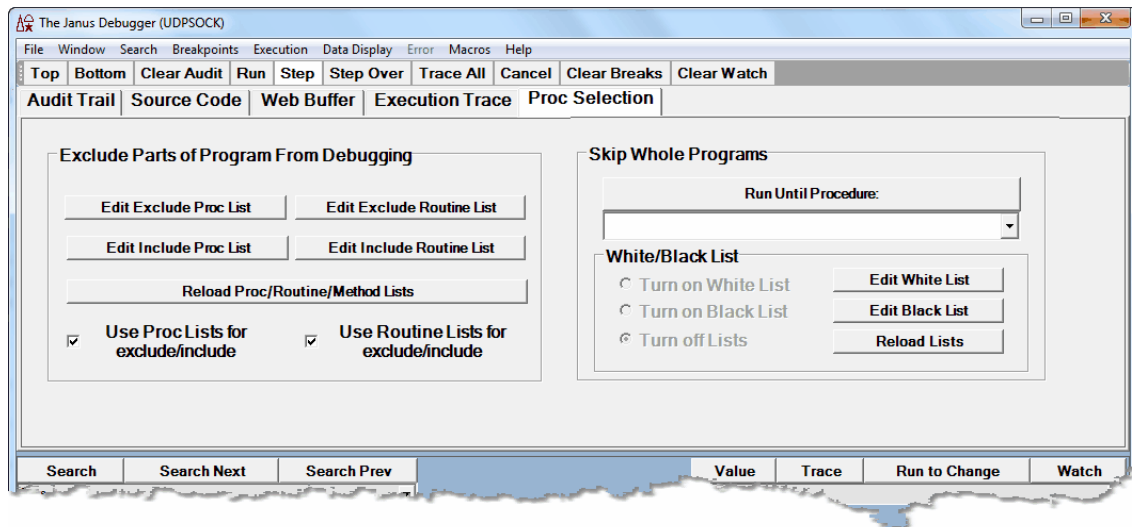
Execution Trace tab

The **Execution Trace** page displays the output of the various tracing operations. It is discussed further in [Tracing program execution](#)^[129].



Proc Selection tab

The **Proc Selection** page lets you designate some sections of your program or entire programs, for debugging while omitting others. This may be for reasons of time or space economy. You can execute but not display in the Debugger Client specified sections of your code, ranging from a few lines to whole routines or inner or outer procedures.



The **Exclude Parts of Program from Debugging** section lets you specify multiple User Language methods, subroutines, or inner procedures whose source code will be executed but not shown in the Debugger Client. From this excluded source code, you can also specify methods, subroutines, or inner procedures whose code you want to be able to view.

The **Skip Whole Programs** section of the page lets you specify by name or name pattern only the outer procedures you want to debug. Those procedures not identified are executed but their code is not sent to or displayed in the Client. You can identify the procedures directly (by explicitly specifying them by name or pattern in a "white list") or indirectly (by explicitly excluding the procedures you do not want to debug by specifying them by name or pattern in a "black list").

The **Proc Selection** page options are discussed further in [Excluding sections of source code from debugging](#).^[66]

See Also

[Performing Basic GUI Tasks](#)^[53]

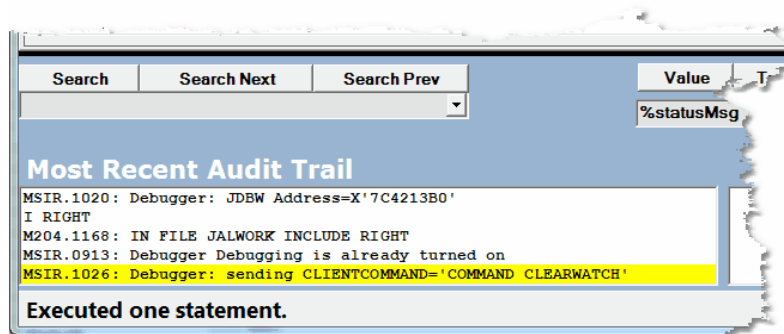
[Opening an external window](#)^[300]

2.2 The lower windows

The windows described here are positioned below the Client's principal display area, the main window.

The Most Recent Audit Trail window

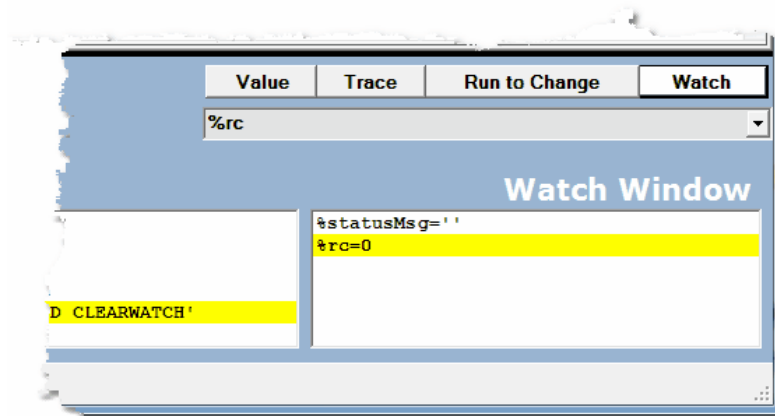
This window displays only the last few lines of the audit trail for this web user or 3270/Batch2 thread. It displays **no** non-audit trail information.



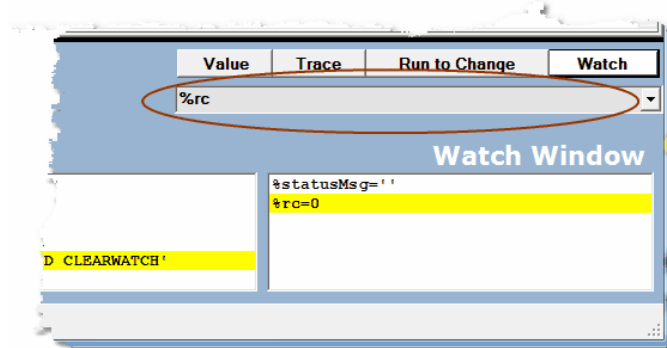
The **Most Recent Audit Trail** display is **not** deleted if you click the **Clear Audit** button in the [button bar](#).^[39]

The Watch Window

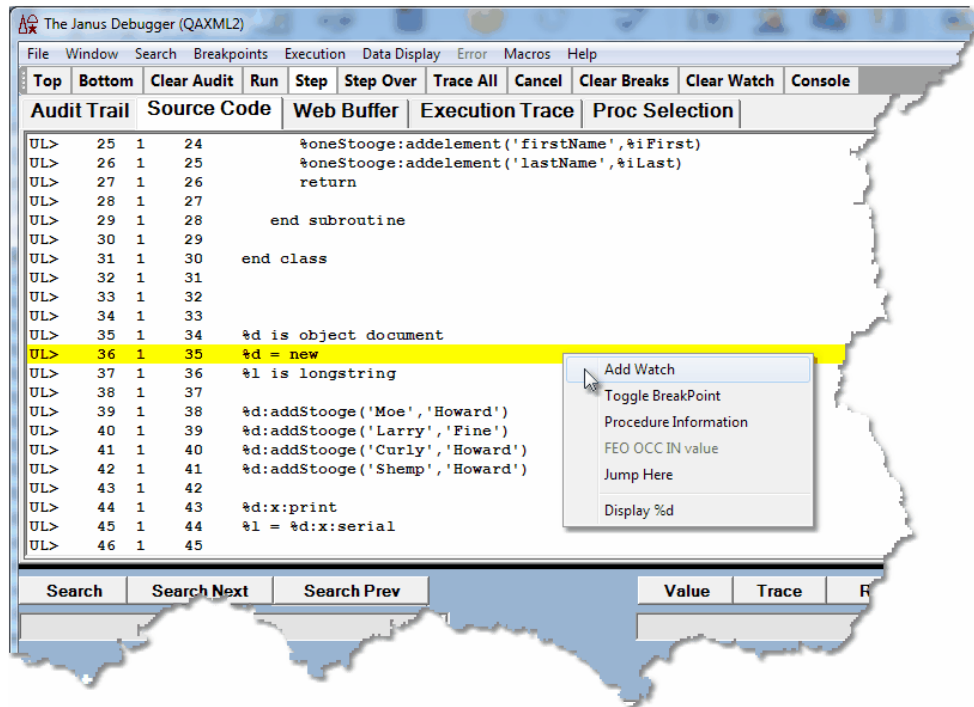
The **Watch Window** box can display the current value of one or more User Language data items (%variables, \$list items, Janus SOAP object variables, etc.). It is updated as the program runs, each time execution is paused by the Debugger:



To add an item to the **Watch Window**, you can use the Entity-name input box below the main window:



Or you can right-click a **Source Code** line and select **Add Watch** from the context menu:



For watched variables that have large values, there are multiple [display options](#)^[90] that include manually widening the the **Watch Window**, viewing the value in a tooltip box, displaying the value in a separate window, or displaying the **Watch Window** itself in a [separate window](#)^[300].

See Also

[Viewing the audit trail](#)^[128]

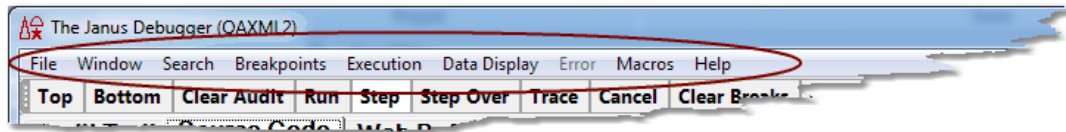
[Watching program data items](#)^[87]

[Opening an external window](#)^[300]

[Hiding the Client's lower windows](#)^[305]

2.3 The menu bar

The Debugger Client has multiple menus, each of which is described in the following subsections. Many of the menu options duplicate the actions of default Client buttons and keyboard shortcuts, and most menu options are associated with Client commands.



If a hot key is mapped to a command that is associated with a menu option, the hot key is displayed in parentheses next to the menu option.

The File menu options [\[18\]](#)

The Window menu options [\[24\]](#)

The Search menu options [\[26\]](#)

The Breakpoints menu options [\[27\]](#)

The Execution menu options [\[29\]](#)

The Data Display menu options [\[33\]](#)

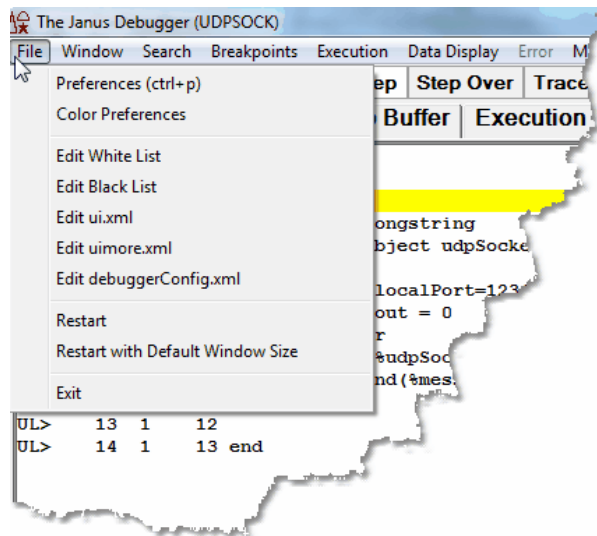
The Error menu options [\[35\]](#)

The Macros menu options [\[36\]](#)

The Help menu options [\[38\]](#)

2.3.1 The File menu options

The File menu options are identified below:



Preferences

Also accessible (by [default](#)^[291]) by using the Ctrl+P keyboard shortcut, this option opens a dialog box that lets you control certain Debugger Client operating options:

☐ Execution Options:

- **Pause at end of evaluation**

Whether to [pause at the end of the evaluation of a request](#)^[61] (to review program data as it is at the end of request processing) before sending any contents of the [web output buffer](#)^[129] or any 3270/Batch2 terminal output.

- **Run Until spans debug sessions**

Whether Client ["Run Until" processing](#)^[75] should continue searching until it finds a specified procedure, even if the program contains HTML frames, the debugging session is interrupted by a loss of the connection to the Model 204 Online, or the Sirius Debugger is toggled off and on again.

- **Break after READ SCREEN**

Whether the Sirius Debugger automatically breaks execution after READ SCREEN or READ MENU statements (see [Breaking at the end of READ SCREEN or READ MENU statements](#)^[62]).

- **Source Preview**

Whether the Client should initially display only a [preview](#)^[85] of the entire source code program (for programs with at least 1000 lines, by default).

- **Macro Autorun**

Whether an include of a procedure from command level will [automatically invoke a macro](#)^[318] whose name matches the name of the included procedure.

☐ *IE Options:*

- **IE Mode**

Whether the Client should automatically configure and maintain [proxy server settings](#)^[383] for the Internet Explorer browser, and whether the proxy is for all host URLs or only for a [specified few](#)^[385].

New in Build 62. Same as the [setIEmode](#)^[258] command.

- **Clear proxy override**

Whether the Client's browser maintenance should clear and preserve any exception URLs [designated in Internet Explorer](#)^[385] to bypass the Client as proxy server.

- **Use existing proxy on not debugged URLs**

Whether the Client should [re-route exception URLs](#)^[385] (designated in Internet Explorer to bypass the proxy server) to a preexisting proxy server rather than directly to the Internet.

☐ *Display Options:*

- **Show at most <x> list items**

What the display maximum is for the number of \$list, Stringlist, or Arraylist object items whose values you can [view in a separate Value window](#)^[104].

- **Restore watches on startup**

Whether to restore this session's remaining Watch Window contents when the Client starts its next session.

- **Trim blanks from selection in View Text**

Whether leading and trailing blanks should be trimmed from selections you copy to the [Text Viewer](#)^[149].

- **History to Execution Trace**

Whether [execution history](#)^[134] data should display in the Execution Trace page instead of a separate window.

- **Show long watch values in a Tooltip**

Whether to display in a tooltip box [Watch Window items](#)^[90] that are too wide to fit within the Watch Window.

- **Use !debugger directives**

Whether to enable [Debugger directives](#)^[67], which let you exclude designated source code from the debugging session.

☐ *Web Server Selection:*

- Which of the Onlines specified in the Client configuration file (`debuggerconfig.xml`) are to have their web requests debugged. For more information, see [Debugging multiple Web Servers](#)^[147].

☐ *Program Titles:*

- **3270 Emulator**

Whether to bring to the top of the PC screen (when the Debugger Client pauses for user to provide external input) the [3270 emulator](#)^[62] or [web browser window](#)^[144] whose title is matched by the text specified in the appropriate one of these boxes.

☐ *Open at Startup:*

Whether the Client should automatically start/restart with:

- The main button bar [in an external window](#)^[42].
- An [extra button bar](#)^[43].
- The **Watch Window**, or the **Audit Trail**, **Web Buffer** or **Execution Trace** page, or a combination of these, [opened in external windows](#)^[305].

☐ *Main Button Bar:*

- **Top, Center, or Bottom**

Whether to [change the position](#)^[41] of the main (non-external) button bar from its default (**Top**), above the main window) to either just below the main window (**Center**), or to the very bottom of the Client window (**Bottom**).

- **Extra Buttons**

Whether [extra buttons](#)^[43] defined in the `ui.xml` file should be added to the display of the main button bar (wherever it is located). If the checkbox is cleared, they display in a separate external window when invoked by menu or command.

New in Build 57. Same as the [extraButtonBar](#)^[207] command with the argument `main`.

☐ *Main Window Options:*

- **Hide Lower Section**

Whether to hide the lower section of the main window (everything below the [tabs](#)^[10]). This is useful in a multiple monitor environment where the **Audit Trail** and **Watch Window** are in separate windows on another monitor.

New in Build 57. Same as the [hideLower](#)^[213] command.

Color Preferences

Lets you [change the color](#)^[293] of text and highlighting in the various Client windows and pages.

Edit White List

Lets you create or edit an existing `whitelist.txt` file. This file contains a list of the *Model 204* procedures that you want to debug.

When [white listing](#)^[79] is activated and the Debugger runs your source code, it filters procedures automatically, stopping to interactively debug only the requests that are on the white list. Other procedures execute normally, but they are not interactively debugged.

Edit Black List

Lets you create or edit an existing `blacklist.txt` file. This file contains a list of the *Model 204* procedures that you want **not** to debug.

When [black listing](#)^[79] is activated and the Debugger runs your source code, it filters procedures automatically, stopping to interactively debug only the requests that are *not* on the black list. Other procedures execute normally, but they are not interactively debugged.

Edit ui.xml

Lets you create or edit an existing [ui.xml file](#)^[287]. This file specifies modifications to the Client's default operational buttons and keyboard shortcuts. You can set the buttons to perform actions (commands), or you can set hot keys to commands.

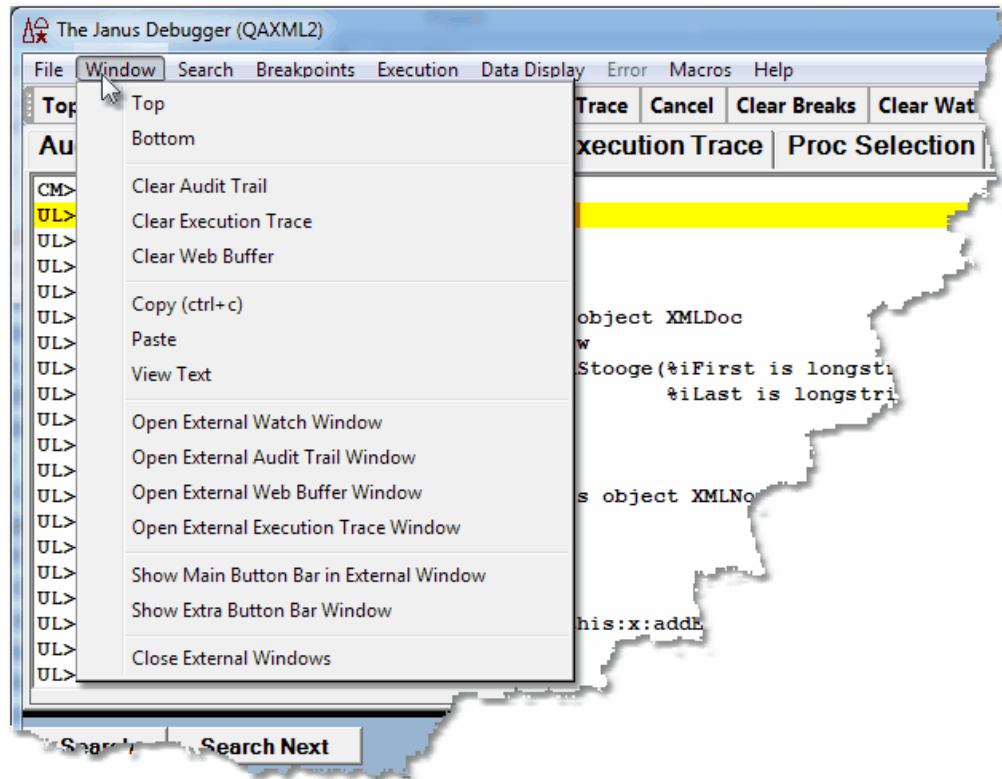
Edit uimore.xml

Lets you create or edit an existing [uimore.xml file](#)^[291]. This file provides the same kind of functionality as, but entirely overrides the `ui.xml` file. New in Build 57.

Edit debuggerConfig.xml	Opens the debuggerConfig.xml ^[374] file for viewing and editing its elements (which define Online connection parameters, file-type filtering, and local editors, among other things).
Restart	Restarts the Debugger Client. Same as the restart ^[242] command.
Restart with Default Window Size	<p>Restarts the Debugger Client with the default size (as when initially installed) for the main window instead of the size at last exit.</p> <p>Same as the restartDefault^[242] command.</p>
Exit	Does no further processing and immediately closes the Client.

2.3.2 The Window menu options

The **Window** menu options are identified below:



Top

Scrolls to the top of the currently displayed tab. Same as the [Top button](#)^[39]. Comparable mappable command, [top](#)^[274], can be applied to the Client window (main, external, work) that you specify.

Bottom

Scrolls to the bottom of the currently displayed tab. Same as the [Bottom button](#)^[39]. Comparable mappable command, [bottom](#)^[183], can be applied to the Client window (main, external, work) that you specify.

Clear Audit Trail

Clears the contents of the **Audit Trail** tab. Same as the [Clear Audit](#)^[128] button and the [clearAudit](#)^[187] command.

Clear Execution Trace	Clears the contents of the Execution Trace tab. Same as the clearExecutionTrace ^[190] command.
Clear Web Buffer	Clears the contents of the Web Buffer tab. Equivalent mappable command is clearWebBuffer ^[193] .
Copy	Copies to the clipboard the lines currently visible in the active tabbed page. Described further in The tabbed pages . ^[10]
Paste	Pastes the current clipboard contents to the Client's Search text box or to the text box above the Watch Window .
View Text	Invokes a separate viewer ^[149] for copying, printing, and saving Client text data.
Open External Watch Window	<p>Displays in an external window^[300], that is, separate from the main window, the current contents of the Client Watch Window. Or, it brings the existing external Watch Window to the top of your current stack of open windows.</p> <p>Same as the openExternalWatchWindow^[233] command.</p>
Open External Audit Trail Window	<p>Displays in an external window^[300] the current contents of the Client Audit Trail tab. Or, it brings the existing external Audit Trail window to the top of your current stack of open windows.</p> <p>Same as the openExternalAuditTrail^[231] command.</p>
Open External Web Buffer Window	<p>Displays in an external window^[300] the current contents of the Client Web Buffer tab. Or, it brings the existing external Web Buffer window to the top of your current stack of open windows.</p> <p>Same as the openExternalWebBuffer^[233] command.</p>
Open External Execution Trace Window	<p>Displays in an external window^[300] the current contents of the Client Execution Trace tab. Or, it brings the existing external Execution Trace window to the top of your current stack of open windows.</p> <p>Same as the openExternalExecutionTrace^[232] command.</p>

Show Main Button Bar In External Window

Displays in an [external window](#)^[300] the current contents of the Client [main button bar](#)^[39]. Or, it brings the existing external **Button Bar** window to the top of your current stack of open windows.

Same as the [buttonBar show](#)^[186] command.

Show Extra Button Bar Window

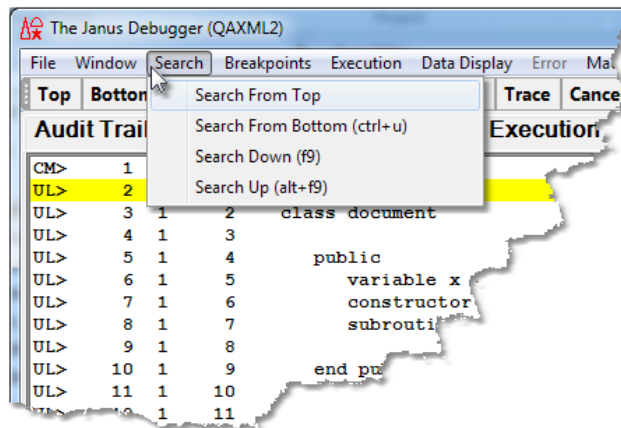
Displays in an [external window](#)^[300] the current contents of the Client [Extra button bar](#)^[43]. Or, it brings the existing external **Extra Buttons** window to the top of your current stack of open windows.

Close External Windows

Closes all Client [external windows](#)^[300]. Same as the [closeExternalWindows](#)^[195] command.

2.3.3 The Search menu options

The **Search** menu options are identified below:



Search From Top

Searches (without regard for case) from the top of the current tab for the string you specify in the **Search** box. Same as the [Search button](#)^[44]. Comparable mappable command, [searchFromTop](#)^[250], can be applied to the Client window (main, external, work) that you specify.

Search From Bottom

Searches (without regard for case) from the bottom of the current tab for the string you specify in the **Search** box. Same as the Alt + **Search** button combination. Comparable mappable command, [searchFromBottom](#)^[248], can be applied to the Client window (main, external, work) that you specify.

Search Down

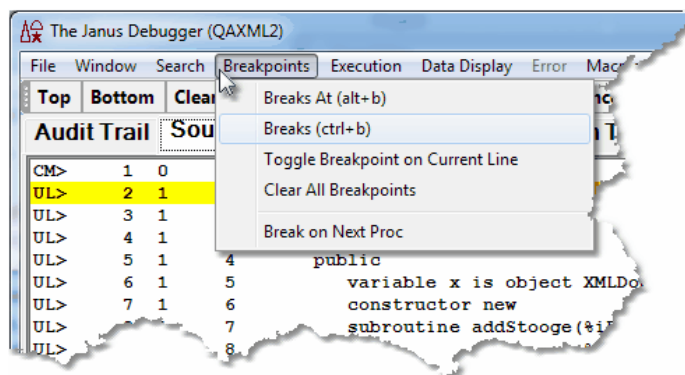
Searches (without regard for case) down (relative to the current line) in the current tab for the string you specify in the **Search** box. Same as the [Search Next button](#)^[44]. Comparable mappable command, [searchDown](#)^[247], can be applied to the Client window (main, external, work) that you specify.

Search Up

Searches up (relative to the current line) in the current tab for the string you specify in the **Search** box. Same as the [Search Prev button](#)^[46] or pressing Alt + **Search Next** button. Comparable mappable command, [searchUp](#)^[251], can be applied to the Client window (main, external, work) that you specify.

2.3.4 The Breakpoints menu options

The **Breakpoints** menu options are identified below:

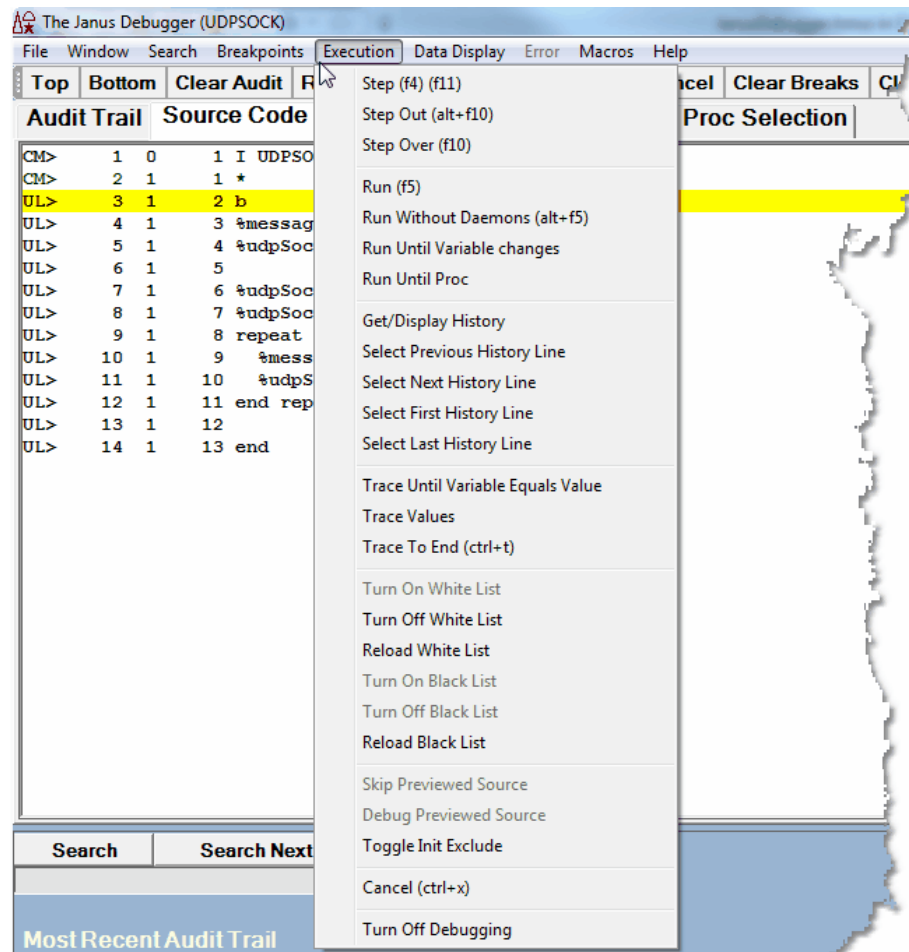
**Breaks At**

Sets a breakpoint on each executable line in the request (from the beginning of the request) that contains a case-insensitive match of the string or regex specified in the **Search** box. Described further in [Setting multiple breakpoints at once](#)^[60].

Breaks	Sets breakpoints on lines that follow comments that have the form <code>*break</code> . Described further in Setting multiple breakpoints at once ^[60] .
Toggle Breakpoint on Current Line	Sets or removes a breakpoint for the current Source Code line if the line is or starts an executable statement. Described further in Setting a single breakpoint ^[58] . Same as the toggleBreakpointOnCurrentLine ^[272] command.
Clear All Breakpoints	Removes all breakpoints in the request. Same as the clearBreaks ^[189] command.

2.3.5 The Execution menu options

The Execution menu options are identified below:



Step Executes the next executable User Language statement.

Step Out Discontinues debugging and leaves the current simple or complex subroutine, user method, or daemon, and resumes debugging on the statement following the statement that called that subroutine, user method, or daemon. Described further in [Stepping out](#)^[64].

Step Over Executes the next executable User Language statement, if it is **not** a simple or complex *Model 204* subroutine or a *Janus SOAP* method. Skips to the statement *after*, if the next executable statement is such a subroutine or method invocation. Same as [using the Step Over button](#)^[56].

Run	Executes User Language statements in the program being debugged until the end of the request or until interrupted by a breakpoint, program error, or sdaemon call.
Run Without Daemons	Performs the same functionality as Run , but unlike Run , is not interrupted by sdaemon code ^[142] .
Run Until Variable Changes	Steps through the program being debugged, stopping if a statement modifies the value of the variable specified in the text box above the Watch Window . Displays the statement that modified the variable and the new variable value in the Execution Trace tab. Same as the Run to Change button ^[133] .
Run Until Proc	Runs program code without interruption until it reaches the procedure specified on the Proc Selection page in the Run Until Procedure text box, then displays that procedure for debugging. Same as the Run to Procedure button ^[75] .
Get/Display History	Displays a history ^[134] of the statements executed <i>thus far</i> during program evaluation. The history includes calls and returns for methods and subroutines (as many as 1000 statements).
Select Previous History Line	<p>Scans chronologically backward in the statement execution history^[134], then highlights <i>in the Source Code or Daemon tab</i> the statement that was executed immediately prior to the statement that is currently highlighted with the Execution Position color^[294].</p> <p>Same as the previousHistory^[237] command.</p>
Select Next History Line	<p>Scans chronologically forward in the statement execution history^[134], then highlights <i>in the Source Code or Daemon tab</i> the statement that was executed immediately following the statement that is currently highlighted with the Execution Position color^[294].</p> <p>Same as the nextHistory^[228] command.</p>
Select First History Line	<p>Scans chronologically backward in the current statement execution history^[134], then highlights <i>in the Source Code or Daemon tab</i> the first (earliest) statement in the history.</p> <p>Same as the firstHistory^[208] command.</p>

Select Last History Line

Scans chronologically forward in the current statement [execution history](#)^[134], then highlights *in the Source Code or Daemon tab* the last (latest) statement in the history.

Same as the [lastHistory](#)^[220] command.

Trace Until Variable Equals Value

Steps through the program being debugged, stopping if a statement modifies the value of the variable specified in the text box above the **Watch Window** so that it equals a value you specify. Displays the statement that modified the variable and the new variable value in the **Execution Trace** tab. Same as [Alt key + Run to Change](#)^[134] button.

Trace Values

Performs the same functionality as **Run**, but also reports in the **Execution Trace** tab all statements that modify a selected variable and what value was assigned to the variable. Described further in [Tracing all updates to a variable's value](#)^[132].

Trace To End

Performs the same functionality as **Run**, but also reports in the **Execution Trace** tab a list of all the statements executed. Described further in [Tracing all lines executed](#)^[130].

Turn On White List

Activates [White List filtering](#)^[81], which by default is not active. Same as clicking the **Turn On White List** button on the **Proc Selection** page or executing the [turnOnWhiteList](#)^[277] command.

Turn Off White List

Deactivates [White List filtering](#)^[81]. Same as clicking the **Turn off Lists** button on the **Proc Selection** page or executing the [turnOffWhiteList](#)^[276] command.

Reload White list

Updates the existing White List with the current contents of the [whitelist.txt file](#)^[79], so you can dynamically update your White List. Same as clicking the **Reload White List** button on the **Proc Selection** page or executing the [reloadWhiteList](#)^[238] command.

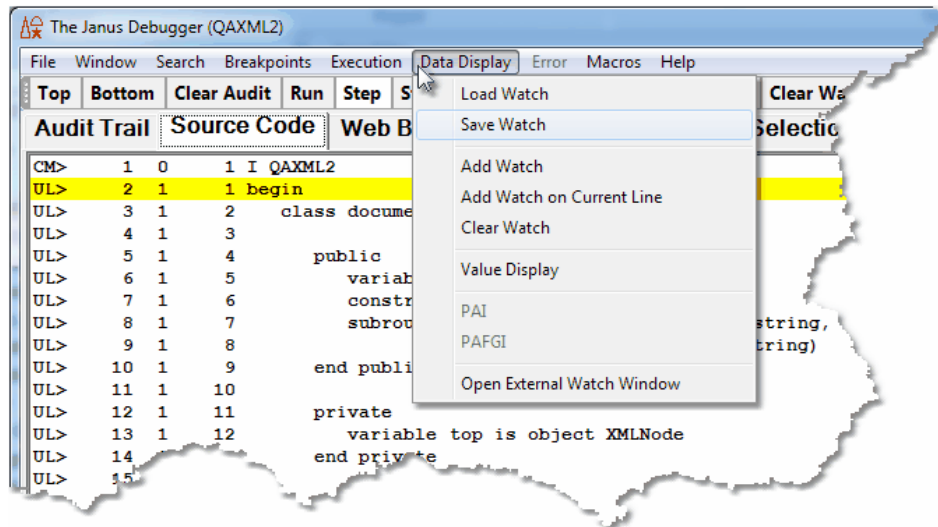
Turn On Black List

Activates [Black List filtering](#)^[81], which by default is not active. Same as clicking the **Turn On Black List** button on the **Proc Selection** page or executing the [turnOnBlackList](#)^[277] command.

Turn Off Black List	Deactivates Black List filtering ^[81] . Same as clicking the Turn off Lists button on the Proc Selection page or executing the turnOffBlackList ^[276] command.
Reload Black list	Updates the existing Black List with the current contents of the blacklist.txt file ^[79] , so you can dynamically update your Black List. Same as clicking the Reload Black List button on the Proc Selection page or executing the reloadBlackList ^[238] command.
Skip Previewed Source	When the Source Preview feature ^[85] is enabled, executes the program that is being previewed but does not download the rest of the source code for viewing or controlled execution.
Debug Previewed Source	When the Source Preview feature ^[85] is enabled, triggers a full download of the program source code for normal debugging. If the program has compilation errors, the full compilation error listing is downloaded.
Toggle Init Exclude	Inverts the way Exclude mode ^[67] operates so that it initially excludes code instead of initially including code (until an explicit directive). Same as toggleInitExclude ^[273] command. Requires at least version 7.6 of the <i>Sirius Mods</i> .
Cancel	Cancels the request being debugged; same as the Cancel button ^[65] . Described further in Cancelling execution ^[65] .
Turn Off Debugging	Stops a debugging session if it is a <i>Sirius Debugger</i> session or if the SIRIUS DEBUG command is being used for a web thread. Same as SIRIUS DEBUG OFF ^[154] command.

2.3.6 The Data Display menu options

The **Data Display** menu options are identified below:



Load Watch

Restores from a local file a list of items to display in the **Watch Window**^[15]. Same as the `loadWatch`^[220] command. For more information, see [Saving and restoring Watch Window contents](#)^[91].

Save Watch

Saves to a local file (for later loading) the list of items currently displayed in the **Watch Window**. Same as the `saveWatch`^[246] command. For more information, see [Saving and restoring Watch Window contents](#)^[91].

Add Watch

Adds to the **Watch Window** the item currently specified in the text area above the **Watch Window**. Same as the [Watch button](#)^[88] or `addWatch`^[179] command.

Add Watch on Current Line

Adds to the **Watch Window** any variables found in the current **Source Code** line. Same as right-clicking the current line and [selecting Add Watch from the context menu](#)^[88], and same as the `addWatchOnCurrentLine`^[180] command.

Clear Watch

Removes all items from the **Watch Window** and instructs the mainframe portion of the Debugger to stop collecting any watch data. Same as the **Clear Watch** button or `clearWatch`^[192] command.

Value Display

Displays in a separate window the details of the value of the item currently specified in the text area above the **Watch Window**. This is one of multiple ways to [display a value](#).^[101] Same as the [valueDisplay](#)^[279] command.

PAI

Displays in a separate window the values of all the visible fields in the current *Model 204* record. This is the output of the User Language PAI (Print All Information) statement. Same as the [pai](#)^[235] command.

Requires at least version 7.6 of the *Sirius Mods* and at least version 7.2 of *Model 204*.

PAFGI

Displays in a separate window the values of all the fields in the current or specified *Model 204* field group. This is the output of the User Language PAFGI (Print All Fieldgroup Information) statement. Same as the [pafgi](#)^[234] command.

Requires at least version 7.6 of the *Sirius Mods* and at least version 7.2 of *Model 204*.

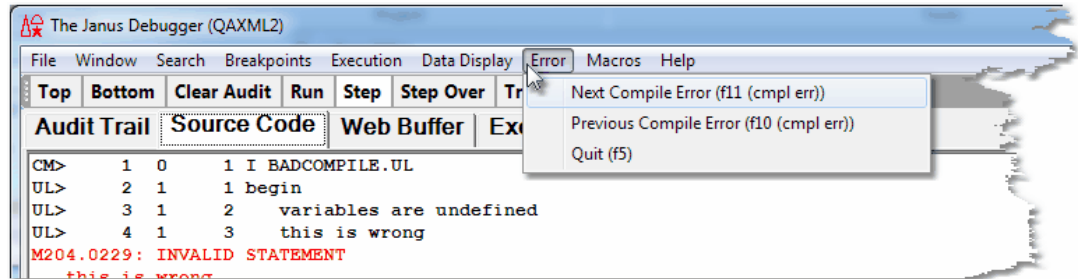
Open External Watch Window

Displays in an [external window](#)^[300] separate from the main Client window the current contents of the Client **Watch Window**. Or, it brings the existing external **Watch Window** to the top of your current stack of open windows.

Same as the [openExternalWatchWindow](#)^[233] command.

2.3.7 The Error menu options

The **Error** menu options are identified below. This menu is *not* enabled if the program you are debugging has no compilation errors.



Next Compile Error

Advances to the next line that has a compilation error, if the request being debugged has more compilation errors after the current one. Described further in [Viewing programs that contain coding errors](#)^[138].

Previous Compile Error

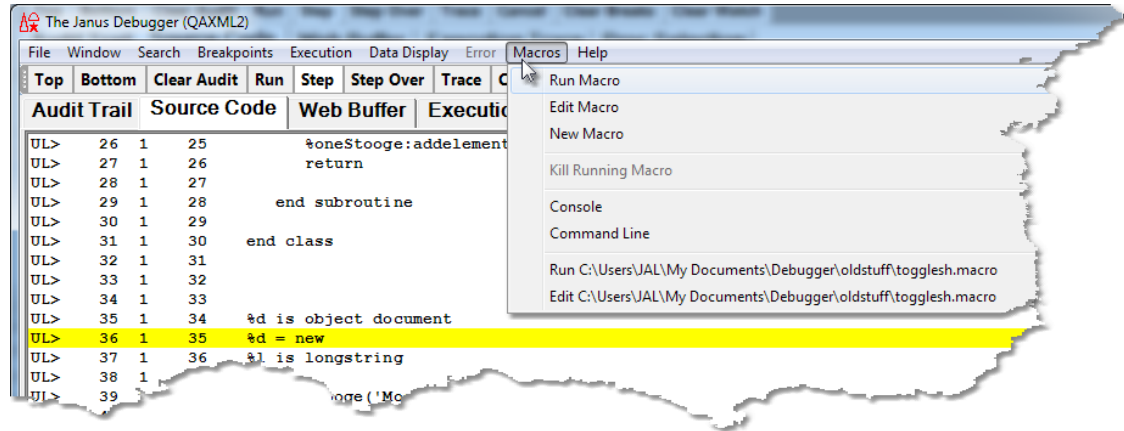
Returns to the previous line that has a compilation error, if the request being debugged has more compilation errors before the current one.

Quit

Stops processing the current request; sends the compilation error messages to the browser (if *Janus Debugger*) or to the terminal (if *Sirius Debugger*).

2.3.8 The Macros menu options

The **Macros** menu options are identified below. Macros are discussed in [Using Debugger macros](#).^[309]



Run Macro

Invokes a Windows file-selection dialog box for you to locate the Debugger macro you want to run.

[runMacroFromUISelection](#)^[244] is the equivalent mappable command.

Edit Macro

Invokes a Windows file-selection dialog box for you to locate the macro you want to modify. Opens the file in the Windows *Notepad* text editor or an alternate editor [specified in the Debugger configuration file](#)^[379].

New Macro

Invokes a Windows file-selection dialog box for you to identify the name and location of the new, blank macro file you are creating. Once you name the file, it is created, then opened for you in the Windows *Notepad* text editor. Equivalent Client command is [createMacro](#)^[200].

This option is labeled **New Blank Macro** in Client builds prior to 53.

Kill Running Macro

Stops the execution of the macro that is running. This can be useful if a macro's execution spans more than one request.

[kill](#)^[218] is the equivalent mappable command.

Console

Invokes a console window that displays information about the macros and commands you run. The console reports the starting and completing of the macro execution, as well as any error messages. Equivalent Client commands are [macroConsole](#)^[221] and [openMacroConsole](#)^[234].

This option is labeled **Macro Console** in Client builds prior to 53.

Command Line

Invokes a dialog box for you to enter the name and any parameters of the macro (or command) you want to run. The macro you identify must be located in the same folder as the Debugger Client executable file or a work folder [known](#)^[296] to the Client.

Equivalent Client command is [openCommandLine](#)^[231].

This option is labeled **Macro Command Line** in Client builds prior to 53.

Run

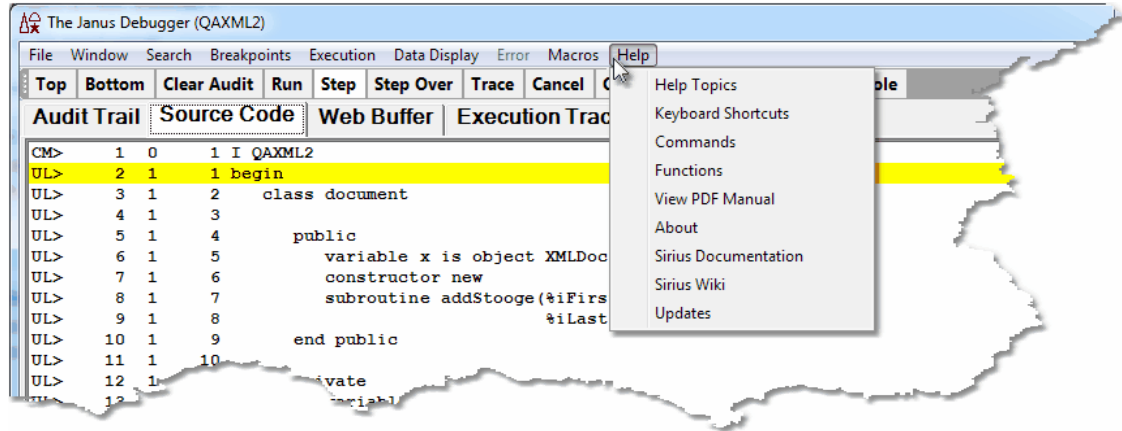
Displays the path to the macro file you last executed. Selecting this item executes the macro again.

Edit

Displays the path to the macro file you last executed. Selecting this item opens the macro for editing in the *Notepad* text editor.

2.3.9 The Help menu options

The Help menu options are identified below:



Help Topics

Displays the Windows online Help contents for the Debugger. Alternative to pressing the F1 key.

Keyboard Shortcuts

Displays the Client's current ([default](#)^[291]) as well as mapped) keyboard shortcuts in the **Keyboard Shortcuts** window. Equivalent mappable command is [showShortcuts](#)^[268].

Commands

Displays a "quick reference" of the set of Client [commands](#)^[179] in the **Commands** window. The commands are listed in alphabetical order and with simple definitions.

Its equivalent mappable command is [showCommands](#)^[266].

Functions

Displays a "quick reference" of the set of [Client functions](#)^[321] in the **&&Functions** window. The &&functions are listed in alphabetical order and include simple definitions. &&functions that may only be used in a macro include a **(macro only)** designation.

New in Client Build 58. Its equivalent mappable command is [showFunctions](#)^[266].

View PDF Manual

Accesses the *Janus/Sirius Debugger User's Guide* (in your Debugger Client installation folder). Its equivalent mappable command is [manual](#)^[224].

About

Displays the contents of the Client's "About box." Its equivalent mappable command is [showAbout](#)^[266].

Sirius Documentation

Opens your browser to the Documentation page of the Sirius Software website ([by default](#)^[379]). This page contains a list of links to the PDF versions of the Sirius manuals.

Sirius Wiki

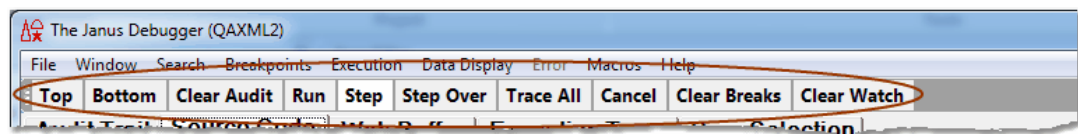
Opens your browser to the Main Page of the Sirius Software wiki for *Model 204* information ([by default](#)^[379]). This wiki contains documentation topics and tutorials for users of *Model 204* and Sirius Software products.

Updates

Invokes a program ([updateGet.exe](#)) from which you can download a new executable file ([JanusDebugger.exe](#)) from a central location to replace your existing Client — if [this feature](#)^[390] has been set up at your site.

2.4 The button bar

Many of the Client's basic operations are controlled by the buttons arrayed in the button bar, which by default is displayed above the main window:



These same operations can also be invoked by Client [menu](#)^[17] options and by keyboard shortcut. The operations are implemented by Client [commands](#)^[200], of which there are many more than there are buttons. You can change the default [mapping of buttons and hot keys](#)^[283] to commands to suit your preference.

The subsections below describe how:

- The buttons may be divided into function categories.
- The [position of the main button bar](#)^[41] may be changed.
- You can [extract the button bar](#)^[42] to a window that is external to the Client.
- You can [create an extra button bar](#)^[43] if you want more than the 15 buttons that the main button bar can accommodate.

Types of buttons

The Client default buttons may be divided by function as follows:

- Page navigation

The **Top** and **Bottom** buttons operate on the currently active [Audit Trail](#)^[10], [Source Code](#)^[11] (or [Daemon](#)^[141]), or [Web Buffer](#)^[12] tab. **Top** brings to the top of the page the first line of the current page. **Bottom** highlights the last line of the current page.

The [mappable Client commands](#)^[285] that perform the same actions are [top](#)^[274] and [bottom.](#)^[183]

- Program execution

The **Step**, **Step Over**, and **Run** buttons advance the **Source Code** page (or **Daemon** page) processing position by executing one or more "executable" User Language statements. For more details, see [Step, Step Over, Run.](#)^[55]

By [default](#)^[291], the F5 key is equivalent to the **Run** button; F4 is equivalent to **Step** (as is F11); and F10 is equivalent to **Step Over**.

The **Cancel** button [stops the execution](#)^[65] of the current program in the **Source Code** page. You can cancel the request at any time.

The mappable Client commands that perform these execution actions are [step](#)^[269], [stepOver](#)^[270], [run](#)^[244], and [cancel.](#)^[187]

- Page clearing

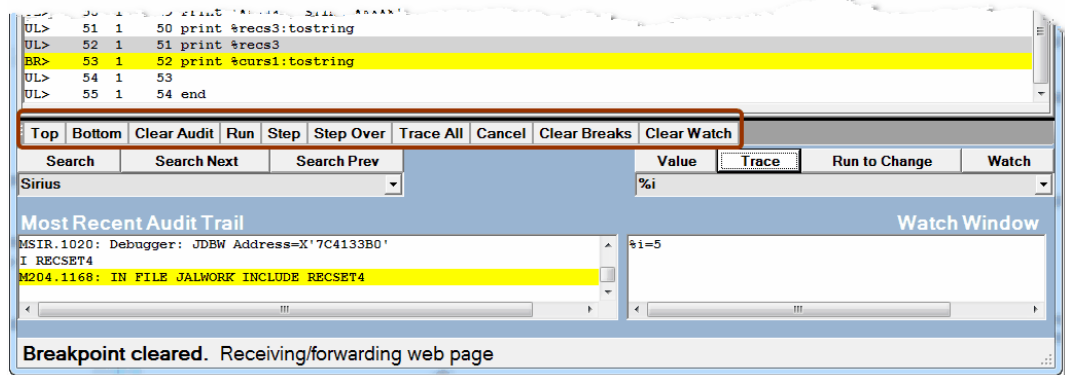
The **Clear Audit**, **Clear Breaks**, and **Clear Watch** buttons remove entirely the contents of the Client's **Audit Trail** tabbed page, the [breakpoints](#)^[57] defined in the program being debugged, and the Client's [Watch Window](#)^[15].

The mappable Client commands that perform these actions are [clearAudit](#)^[187], [clearBreaks](#)^[189], and [clearWatch](#)^[192].^[187]

Positioning the button bar within the Client window

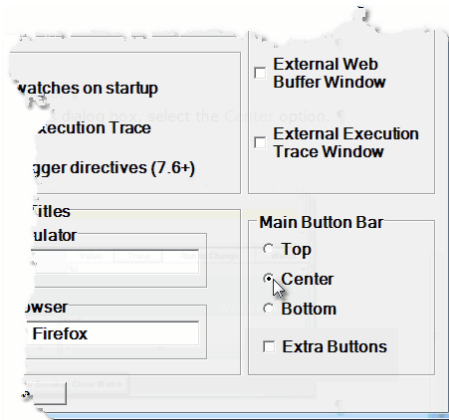
You can change the location of the main button bar from its default (above the main tabs) to either of these other positions in the Client window:

- Immediately below the main tabs (but above the search, tracing, and value displaying controls):



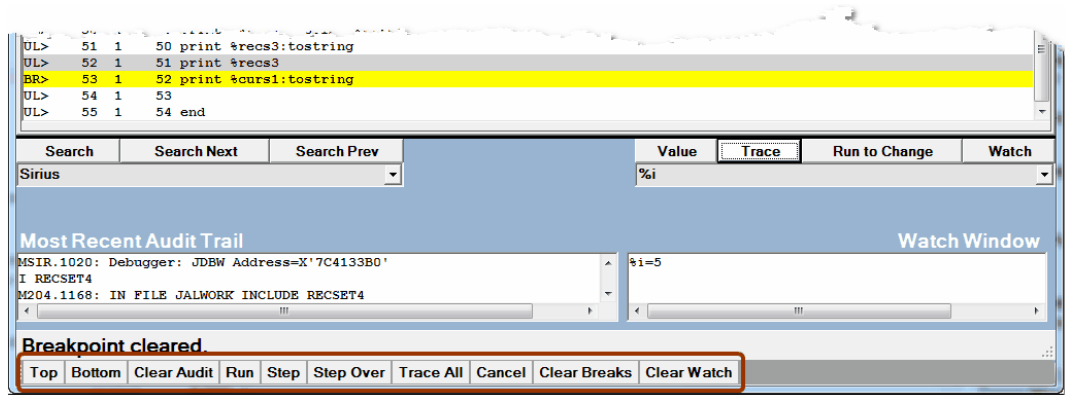
To move the button bar to this position:

1. From the Client's **File** menu, select **Preferences**, or use the Ctl-P keyboard shortcut or the [preferences](#) ²³⁶ command.
2. In the **Main Button Bar** area of the **Preferences** dialog box, select the **Center** option:



The button bar immediately relocates.

- At the very bottom of the Client window:



To move the button bar to this position, select the **Bottom** option in the **Main Button Bar** area of the **Preferences** dialog box.

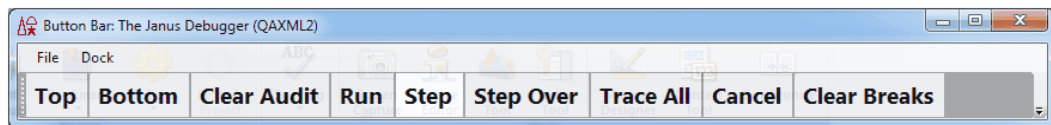
The [mappable Client command](#)^[285] that positions the button bar is [mainButtonBar](#)^[223]

Launching an external button bar

For your convenience, the location of the button bar is not fixed. In addition to changing its position on the main Client window as described above, you can move the main button bar to a separate window outside the Client application window. You do so in either of the following ways:

- Selecting the **Show Main Button Bar in External Window** option of the **Window** menu (prior to Client Build 56, this is the **Open External Button Window** option)
- Invoking a button, key, or macro [mapped](#)^[283] to the [openExternalButtonWindow](#)^[232] or the [buttonBar](#)^[186] command

The resulting external **Button Bar** window shares the characteristics of the other Client [external windows](#)^[300]. You can close the external window at any time by Client command ([closeExternalButtonWindow](#)^[194]), by the **Exit** option of the button bar **File** menu, or simply by clicking the **X** button in the upper-right corner. In addition, the opened **Button Bar** window has **Dock** menu options, as described below.



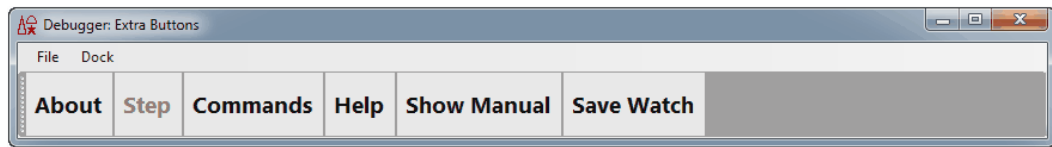
The `buttonBar` command has parameter options that let you select where on your desktop to locate the window. You can dock it, stationary, in the top or bottom left corner of the desktop, or you can simply open it, undocked, as with the launching options (menu and command) described above. Also, once opened, the **Button Bar** window **Dock** menu has **Top**, **Bottom**, and **Float** options that perform the same functions as the corresponding `buttonBar` command arguments.

Launching a second button bar

As of Client Build 56, you may open an additional button bar (in the **Extra Buttons** external window) which can contain as many as 15 buttons. Like the buttons in the main button bar, these extra buttons are `mappable`^[285] in the `ui.xml` file or via the `mapButton`^[224] command. You launch such an extra button bar by either of the following:

- Selecting the **Show Extra Button Bar Window** option of the **Window** menu
- Invoking a button, key, or macro `mapped`^[283] to the `extraButtonBar`^[207] command

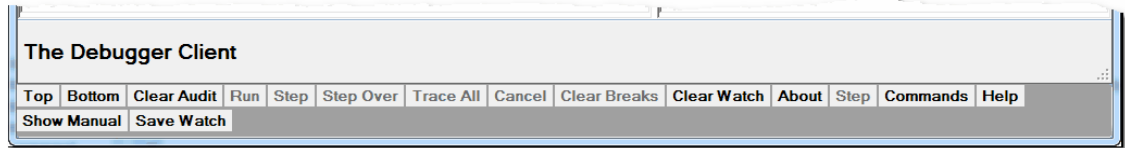
The resulting external **Extra Buttons** window is just like the external main **Button Bar** window (described above), and it shares the characteristics of the other Client [external windows](#).^[300]



As stated above, you add buttons to the extra button bar by mapping them in the `ui.xml` file to the commands you want them to execute. The mapping commands are the same as those for the buttons in the main button bar, except the buttons are `named`^[285] `extraButton0` through `extraButton14`. For example, the buttons in the extra button bar shown above result from the following commands:

```
<mapping command="showAbout" button="extrabutton0" />
<mapping command="step" button="extrabutton1" />
<mapping command="showCommands" button="extrabutton2" />
<mapping command="help" button="extrabutton3" />
<mapping command="manual" button="extrabutton4" />
<mapping command="saveWatch" button="extrabutton5" />
```

As of Build 57, you may add your extra buttons to the existing main button bar instead of opening an additional window. For example, the following image shows the extra buttons (defined above) added to the main button bar, which is positioned at the bottom of the Client main window:



To merge extra buttons with those in the main button bar (instead of opening a second button bar), define the mapping commands in `ui.xml` as above, then do either of the following:

- Select the **Extra Buttons** checkbox of the [Preferences](#)^[18] dialog box
- Invoke a button, key, or macro mapped to an [extraButtonBar](#)^[207] command that specifies the position parameter `main`.

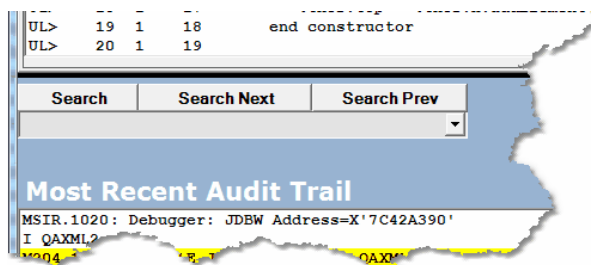
If the main and second button bars are open in separate external windows, selecting the **Extra Buttons** checkbox or invoking an `extraButtonBar main` command merges all the buttons in the primary external button bar.

Once a merged button bar exists, the extra buttons remain part of the primary button bar if it is subsequently moved to an external window from the Client, or if moved back to the Client from an external window.

To decouple a merged button bar, you can clear the **Extra Buttons** checkbox. Less directly, you can select **Window > Show Extra Button Bar Window**; this action simultaneously clears the **Extra Buttons** checkbox.

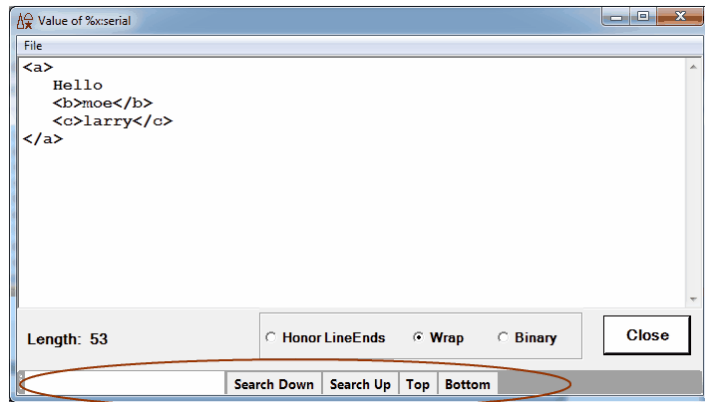
2.5 The search facility

The Client search facility is available for its main pages as well as for its many Value and work windows. The most handy search controls are the **Search**, **Search Next**, and **Search Prev** buttons that operate on the currently active page.



Comparable controls are available, alternatively, from the [Search menu](#)^[26] and its analogous Client [commands](#)^[285], and from the multiple hot key and key combinations (described below) that also perform these search functions.

These search functions are also provided from an on-window search bar when you choose to view the main pages as [external windows](#)^[300] or when you invoke a Client **Value** or other work window ([Console](#)^[316], [history](#)^[134], etc.):



The search functions on these windows are subject to Client command control via an **In window** prefix for the individual search command ([searchDown](#)^[247], [searchFromBottom](#)^[248], [searchFromTop](#)^[250], [searchUp](#)^[251], [top](#)^[274], [bottom](#)^[183]).

The Search button

Clicking the **Search** button on the main window searches the current tab from its top (first) line for the string you specify in the text box above (without regard for case).

To repeat a search from the current line, you use the **Search Next** button. As an alternative, you can also click (give focus to) the Search text box below or use the Ctrl+F key combination (which performs the same function, by [default](#)^[291]), then press the Enter key.

To search from the last line of the tab toward its first, you press the Alt key while clicking the **Search** button (or use the Ctrl+U key combination, which by default performs the same function). You can then use the [Search Prev](#)^[46] button to search backwards for the next occurrence of the search string.

To repeat a search for a previous search string, click the arrow button to the right of the search box for a history of as many as twenty previous search terms.

To use regular expressions in the search string, begin the string with a tilde (~). For example, the following string matches variable declarations of format **%name IS whatever:**

```
~%\w+\s+IS
```

Note: The rules for VB .net regular expressions are observed; these differ from the rules for Sirius User Language regular expressions. For background information about regular expressions, a good resource is *Mastering Regular Expressions*, by Jeffrey E. F. Friedl, published by O'Reilly Media, Inc. (2nd edition, July 15, 2002).

To use the value of a macro variable or Client function, both of which begin with an ampersand (&), in the search string, simply specify the name as is. To search for a string that starts with an ampersand and is not a macro variable or function name, prefix the initial ampersand with a backslash character (\) to treat the the string as a literal. Similarly, specify \\ to search for a single backslash character. This backslash escape character is valid as of Client Build 58.

The **Search** button has the same effect as the [Search From Top](#)^[26] option in the **Search** menu and as the [searchFromTop](#)^[250] command. The Alt + **Search** button has the same effect as the [Search From Bottom](#)^[26] option in the **Search** menu and as the [searchFromBottom](#)^[248] command.

The Search Next button

Clicking the **Search Next** button starts from the current position and searches the current tab for the string you specify, or it repeats the previous forward search. The F9 key performs the same function (by [default](#)^[291]). And pressing the Enter key after clicking **Search Next** (or whenever the **Search Next** button is highlighted) repeats the **Search Next** action.

If you press the Alt key while clicking the **Search Next** button (or press Alt+F9), you search again for the current search string, but the search is backwards, towards the top from the current position. This is the same as using the **Search Prev** button, as described below.

The **Search Next** button has the same effect as the [Search Down](#)^[26] option in the **Search** menu, as the **Search Down** button in Client external windows, and as the [searchDown](#)^[247] command.

The Search Prev button

Clicking the **Search Prev** button starts from the current position and searches backwards (towards the top) in the current tab for an occurrence of the currently specified search string. Pressing the Enter key after clicking **Search Prev** (or whenever the **Search Prev** button is highlighted) repeats the **Search Prev** action.

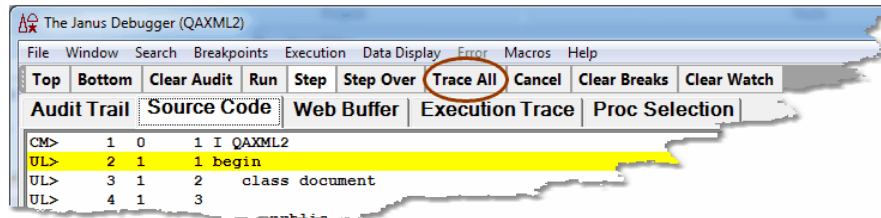
Alternatives that have the same affect as the **Search Prev** button are:

- Pressing the Alt key while clicking the **Search Next** button
- Pressing the Alt+F9 key combination, which (by [default](#)^[291]) performs the same function

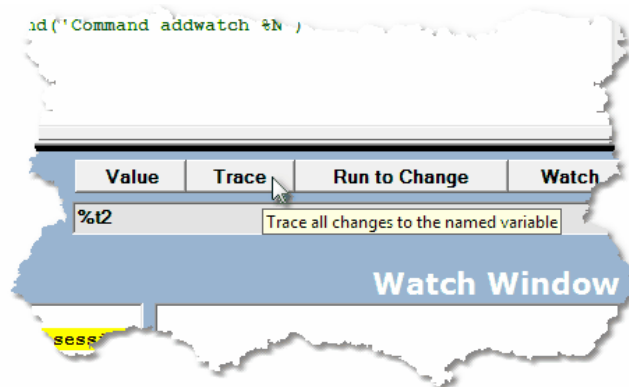
The **Search Prev** button has the same effect as the [Search Up](#)^[26] option in the **Search** menu, as the **Search Up** button in Client external windows, and as the [searchUp](#)^[251] command.

2.6 The tracing options

The **Trace All** control button lets you trace from the current program point to the end of the program, displaying a list of all executed statements. The Ctrl+T keyboard shortcut is equivalent (by [default](#)^[291]) to clicking the **Trace** button:



The **Trace** button below the main window lets you trace all statements that modify a variable you specify, also displaying what value was assigned to the variable:



The **Run to Change** button stops program execution if the variable you specify is modified:

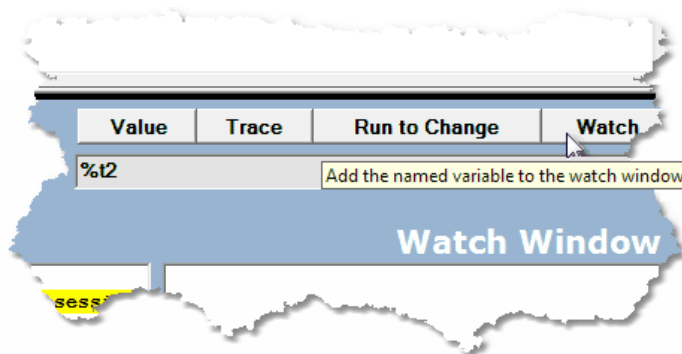


See Also

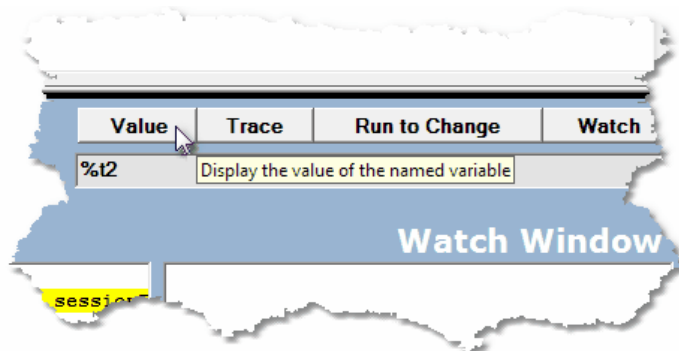
[Tracing execution](#)^[129]

2.7 The value displaying controls

The **Watch** button adds to the **Watch Window** the item you specify in the [Entity-name input box](#)^[51] below the button:



The **Value** button displays in a separate window the value of the item you specify in the Entity-name input box:

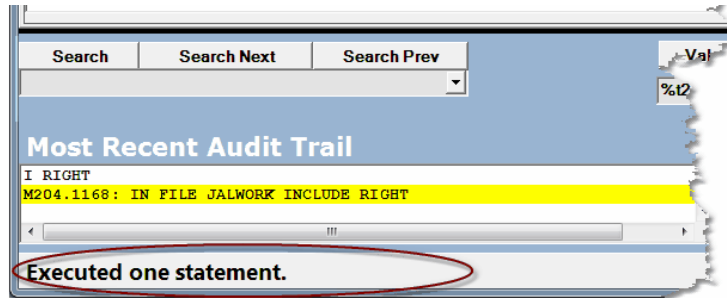
**See Also**

[Watching program data items](#)^[87]

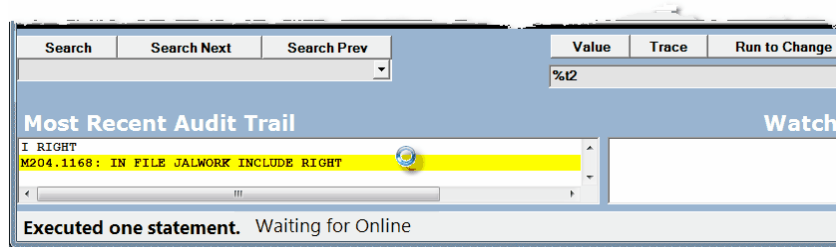
[Displaying temporarily the value of a program data item](#)^[101]

2.8 The Status bar

The Status bar in the Client's bottom left corner displays the state of the Debugger after each operation you invoke:

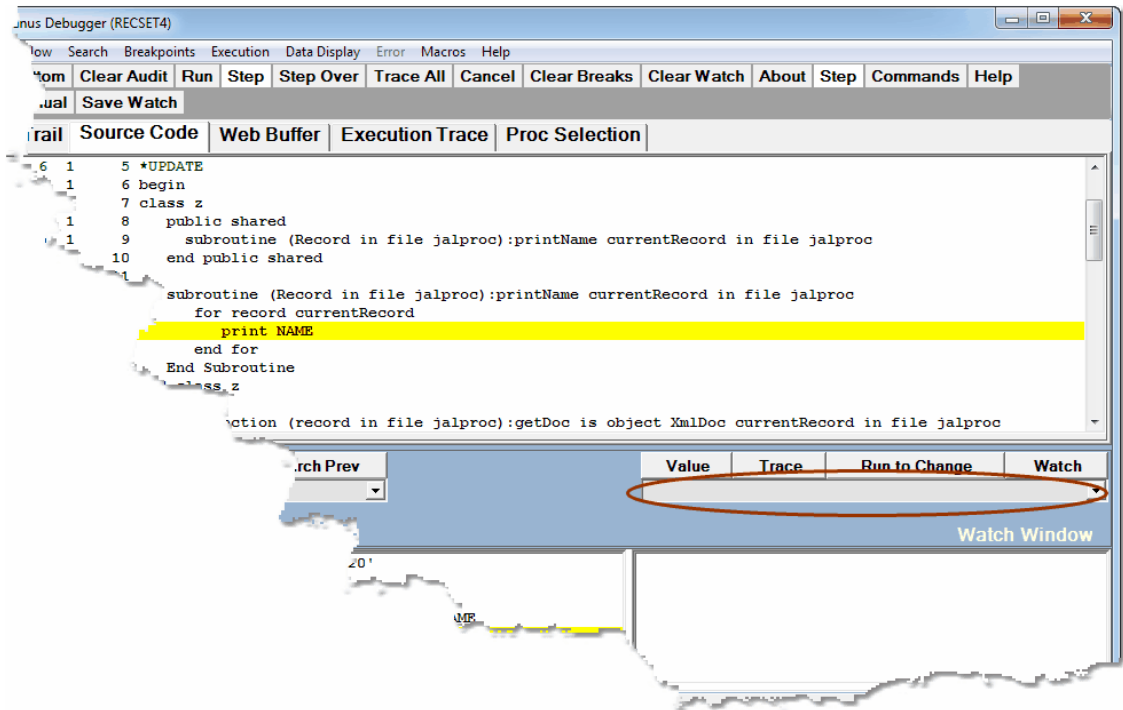


"Waiting" messages (say, for lengthy online processing or network transmissions) are also displayed along with the operational messages:



2.9 The Entity-name input box

This text box, located below the Client's main window, is used for supplying the name of any of a variety of program code entities (%variable or field, for example) to be variously traced, watched, displayed, expanded, and so on:



CHAPTER 3 ***Performing Basic GUI Tasks***

These sections describe how you perform basic operations using the Debugger:

[Controlling the execution of program code](#)^[54]

[Viewing and modifying program elements](#)^[87]

[Getting source file, audit trail, and web buffer information](#)^[126]

[Tracing program execution](#)^[129]

[Viewing programs that contain coding errors](#)^[138]

[Debugging requests that spawn daemons](#)^[141]

[Debugging Web Server persistent sessions](#)^[144]

[Debugging multiple Web Servers](#)^[147]

3.1 Controlling the execution of program code

This section describes how to navigate through and control the execution of the program code you are debugging.

GUI buttons, menus, and commands let you execute a User Language request all at once, or let you advance execution in increments (by pausing after executing individual statements, optionally bypassing the code display from calls to subroutines and methods).

You can explicitly set "breakpoints" in a request before which program execution is to pause, and you can toggle defaults a) to pause or not at the end of the evaluation of a request, and b) to pause or not after processing READ SCREEN statements.

You can discontinue the debugging of all or part of a request called by your program, skipping the execution of the entire request or of the remaining part of the request.

You can continue code execution without interruption until a procedure you identify by name or pattern is reached. And you can jump out of the normal flow of code execution, executing the statement to which you jump, but not executing the intervening statements.

These subsections follow:

[Step, Step Over, and Run](#)^[55]

[Using breakpoints](#)^[57]

[Suppressing the break at the end of request evaluation](#)^[61]

[Breaking after READ SCREEN or READ MENU statements](#)^[62]

[Stepping out](#)^[64]

[Cancelling execution](#)^[65]

[Excluding sections of source code from debugging](#)^[66]

[Altering the flow of execution](#)^[83]

[Previewing program code](#)^[85]

3.1.1 Step, Step Over, and Run

These three action options (activated by same-named control buttons, commands, or menu item) advance the **Source Code** page (or [Daemon page](#)^[141]) processing position by executing one or more "executable" User Language statements:



A statement is executable if it produces a run-time action. Neither a variable declaration statement nor a *Janus SOAP* class definition statement, for example, is executable in this sense.

When no executable code operations remain, these buttons are dimmed.

When a button is highlighted (color is white, border is bold), the action it invokes can be executed by pressing the Enter key.

Step

Step executes a single User Language statement. When the statement has executed:

- **Executed one statement** is displayed in the [Status box](#)^[50].
- The next line to be executed is highlighted.

If you click **Step** and the current statement (a subroutine call, for example) invokes other statements in the program, the Debugger first "executes" the call statement itself by moving to and highlighting the first of the executable subroutine statements. With each subsequent click of **Step**, the Client steps through the subroutine, executing one statement at a time.

If you are at the end of a request (the **End** statement is highlighted) and click **Step**, **Evaluation successfully completed** is displayed in the status box, and execution pauses ([by default](#)^[61]), giving you a final review. If you click **Run**, the Debugger Client sends any contents of the [web output buffer](#)^[129] to the browser, or it sends any 3270/ Batch2 terminal output to the terminal; then it advances execution to the next request, if any more requests are queued.

Note: Pressing the F4 key or the F11 key is the same as clicking **Step** button (unless you have [reconfigured](#)^[283] your hot keys).

Pressing the Enter key after clicking **Step** (or whenever the **Step** button is highlighted with a white background) repeats the **Step** action.

The other **Step** button equivalents are the [step](#)^[269] command and the **Step** option of the **Execution** menu.

Step Over

Step Over functions like **Step** with one important difference: it skips subroutines and methods. If the execution position is immediately before an invocation of a *Janus SOAP* method or a simple or complex *Model 204* subroutine, clicking **Step Over** advances the execution position to immediately before the statement **after** the subroutine or method invocation. No debugging is done in the stepped-over subroutine or method, nor in any code or daemons that it might call.

Step Over is useful if you know a particular subroutine or method works and you do not want to interactively execute it.

Note: Pressing the F10 key is the same as clicking the **Step Over** button (unless you have [reconfigured](#)^[283] your hot keys).

Pressing the Enter key after clicking **Step Over** (or whenever the **Step Over** button is highlighted with a white background) repeats the **Step Over** action.

Pressing the Alt key while clicking **Step Over** invokes a [Step Out](#)^[64]. Pressing the Alt+F10 key combination has the same effect.

The other **Step Over** button equivalents are the [stepOver](#)^[270] command and the **Step Over** option of the **Execution** menu.

Run

The **Run** button “resumes execution” of the program. The User Language statements execute normally, until one of the following events occurs:

- End of request (the final end statement is highlighted, and **Evaluation successfully completed** is displayed in the status box)
- A cancelling error such as subscript out of range, or a null object reference (the line that raised the error is highlighted)
- A [breakpoint](#)^[57] (the line containing the breakpoint is highlighted)
- Code that an sdaemon executes is called

Once the end of a request is reached, execution pauses ([by default](#)^[61]), giving you a final review. You must click **Run** again for the Debugger to send any contents of the [web output buffer](#)^[129] to the browser or any 3270/Batch2 terminal output to the terminal, and advance execution to the next request, if any more requests are queued.

If no further requests are queued, the **Run** button is disabled.

Note: Pressing the F5 key is the same as clicking the **Run** button (unless you have [reconfigured](#)^[283] your hot keys).

Pressing the Enter key after clicking **Run** (or whenever the **Run** button is highlighted with a white background) repeats the **Run** action.

The other **Run** button equivalents are the [run](#)^[244] command and the **Run** option of the **Execution** menu.

3.1.2 Using breakpoints

If you set a **breakpoint** on a line on the **Source Code** tab (or [Daemon tab](#)^[141]), then run the program, program execution is paused just before that line is to be executed (if that line is to be executed). When you set a breakpoint:

- Execution will be paused immediately before the execution of the line for which the breakpoint is set.
- The line with the breakpoint is highlighted.
- **Breakpoint set** is displayed in the status bar.

These subtopics follow:

[Setting a single breakpoint](#)^[58]

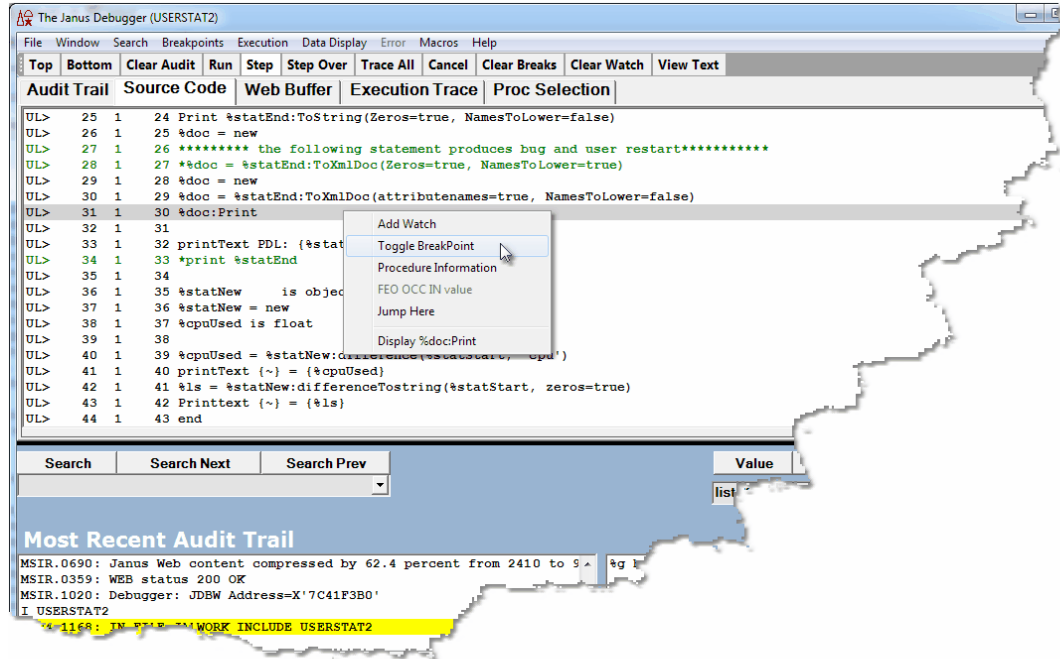
[Setting multiple breakpoints at once](#)^[60]

[Clearing a breakpoint](#)^[60]

[Clearing all breakpoints](#)^[61]

Setting a single breakpoint

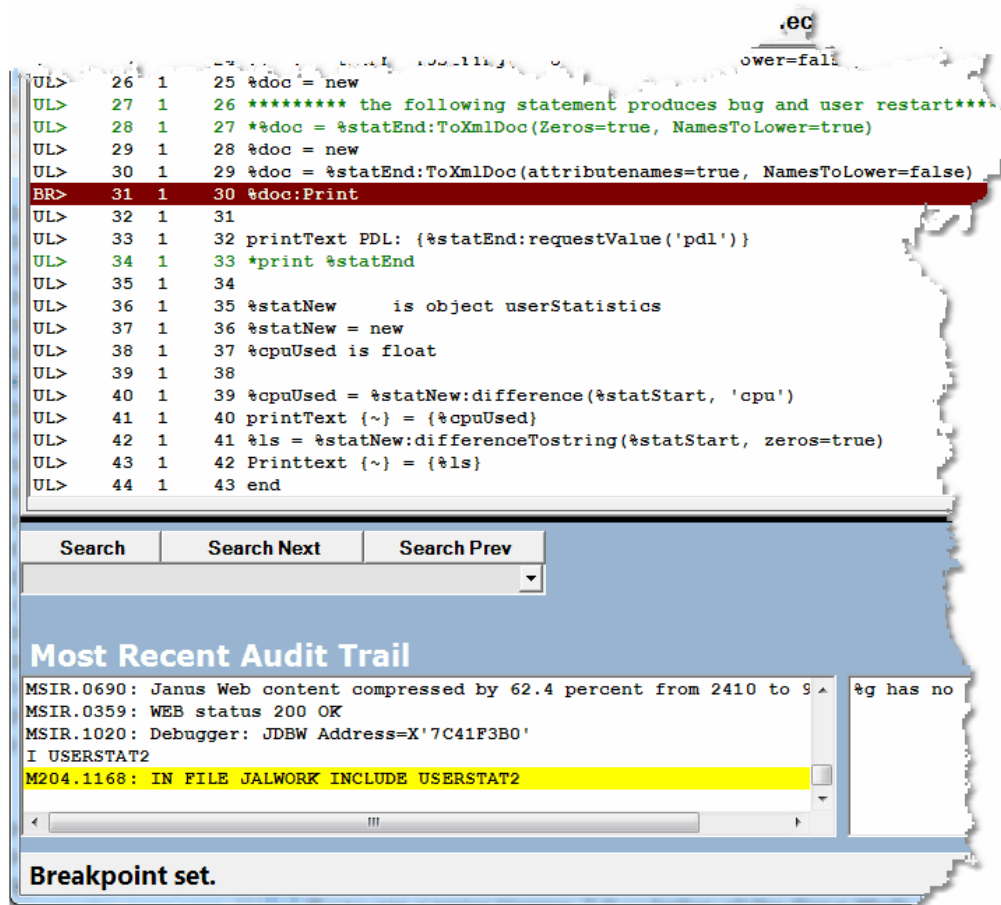
To set a breakpoint on a line, you can simply double-click the desired line. Alternatively, you can right-click the line and select **Toggle BreakPoint** from the context menu:



Other ways to set this breakpoint are:

- Select **Toggle Breakpoint on Current Line** from the **Breakpoints** menu.
- Use the [setBreakpointOnCurrentLine](#)^[257] or the [toggleBreakpointOn](#)^[272] [mappable](#)^[285] Client command.

Once a breakpoint is set, the **UL>** at the beginning of the source code line changes to **BR>**, the highlight color changes, **Breakpoint set** is displayed in the [status bar](#)^[50], and the Debugger Client is made the topmost window on the browser screen:



Only executable statements may be breakpoints: if you try to make a non-executable statement a breakpoint, the Debugger sets the breakpoint on the next executable statement below the line you selected.

You can set as many as 1000 breakpoints in a single User Language request. Attempting to set more than the maximum is not allowed (results in an error message display).

Setting multiple breakpoints at once

You can set multiple breakpoints at once, using either of two approaches:

- With a search string, set breakpoints on all matching lines
- With a Ctrl+B keystroke (if using the [default assignments](#)^[291]), set breakpoints on executable statements that follow `"*Break"` comment lines in the source code

Using a search string:

1. In the [Search text box](#)^[39], specify a search string or a [regular expression](#)^[45] (regex).
2. Press the Alt+B key combination (if using the default assignments), or use the **Breakpoints** menu **Breaks At** option.

A breakpoint is set on each executable line in the request (from the beginning of the request) that contains a case-insensitive match of the string or regex.

No Breakpoints are set in code that is not currently displayed in the **Source Code** page (for example, in daemons called by the current request).

Your current execution point in the request is not affected.

The mappable Client command that sets multiple breakpoints based on a search string is [breaksAt.](#)^[185]

Using Ctrl+B:

When you click the Ctrl+B key combination (if using the default assignments), or when you use the **Breakpoints** menu **Breaks** option, the Debugger Client scans the current User Language request from the beginning to the end of the request for lines beginning with the string `*Break`. Whenever such a line is found, a breakpoint is set on the next line if it is an executable statement.

No breakpoint is set after an occurrence of `*Break` unless the following is true:

- `*Break` (case not important, but no intervening blanks allowed) must be the first non-blank characters on the line. Any other characters may follow.
- The line immediately following the `*Break` comment must be an executable statement.

The mappable Client command that sets multiple breakpoints based on `*Break` is [breaks.](#)^[184]

Clearing a breakpoint

To remove a single breakpoint, do any of the following:

- Double-click the line.

- Right-click the line and select **Toggle Breakpoint**.
- Select **Toggle Breakpoint on Current Line** from the **Breakpoints** menu.
- Use the [clearBreakpointOnCurrentLine](#)^[188] or the [toggleBreakpointOnCurrentLine](#)^[272] mappable Client command.

As a result, the **BR>** indicator changes back to **UL>** to indicate successful removal, the highlight color changes, and **Breakpoint cleared** is displayed in the status box.

Clearing all breakpoints

To clear all the breakpoints that are set, click the **Clear Breaks** button or use the **Breakpoints** menu **Clear All Breakpoints** option. All breakpoints get cleared, and all breakpoint indicators in the source display are changed back.

The mappable Client command that clears all breakpoints is [clearBreaks](#).^[189]

See Also

[Break method](#)^[162]

[Running to a specific procedure](#)^[75]

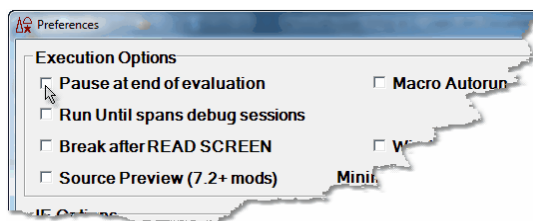
[Stepping out](#)^[64]

3.1.3 Suppressing the break at the end of request evaluation

By default, the Debugger Client pauses at the end of the evaluation of a request before it sends any contents of the [web output buffer](#)^[129] or any 3270/Batch2 terminal output. This lets you review program data as it is at the end of request processing.

If you prefer to have processing continue without stopping at this point, you can suppress the pause, as follows:

1. In the Client, select **Preferences** from the **File** menu (or use the Ctrl+P keyboard shortcut).
2. In the **Preferences** dialog box, clear the **Pause at end of evaluation** checkbox, then click **Done**:



Pausing can be restored at any time by selecting the checkbox again.

Note: The Client [setPreference](#)^[260] command has an option that lets you toggle the **Pause at end of evaluation** checkbox.

The **Pause at end of evaluation** setting that exists at the end of the Debugger Client session persists to the next run of the Client.

3.1.4 Breaking after READ SCREEN or READ MENU statements

Normally, the Debugger Client for the *Sirius Debugger* pauses at a READ SCREEN or READ MENU statement (displaying a **Full Screen Read Pending** message in the [Status bar](#)^[50]), waiting for input from the Online user. Once you complete the input, the Client evaluates your response, displays a **READ SCREEN Completed** message in the Status bar, and continues (without pause, by default) processing statements until it reaches a breakpoint or the end of the request.

These events are reported in a sequence of lines in the Client **Audit Trail** page like the following:

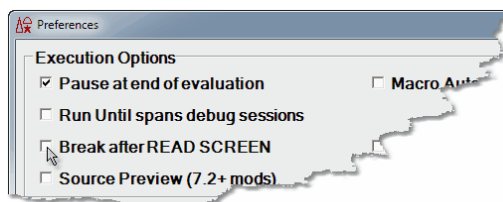
```
... 10:01:48.27      2      20 LI I SCREENO
... 10:01:48.27      2      20 MS M204.1168: IN FILE GWDEB INCLUDE SCREENO
... 10:02:10      Full Screen Read Pending
... 10:02:22      READ SCREEN completed
```

If you want to examine how the program handles the user response to the READ SCREEN or READ MENU, for example, you can do either of the following:

- Explicitly [set a breakpoint](#)^[58] after the READ SCREEN or READ MENU statement.
- Have processing automatically paused by default *after* the user interaction following READ SCREEN or READ MENU.

To invoke the second of the preceding options:

1. In the Client, select **Preferences** from the **File** menu (or use the Ctrl+P keyboard shortcut).
2. In the **Preferences** dialog box, select the **Break after READ SCREEN** checkbox (it is clear by default), then click **Done**:



After the Online user replies to the next READ SCREEN or READ MENU statement, the Debugger will pause at the statement following the READ SCREEN or READ MENU, and **READ SCREEN completed** will display in the Status bar.

Running without a break after replying to READ SCREEN or READ MENU can be restored at any time by clearing the **Break after READ SCREEN** checkbox.

Note: The Client [setPreference](#)^[260] command has an option that lets you toggle the **Break after READ SCREEN** checkbox.

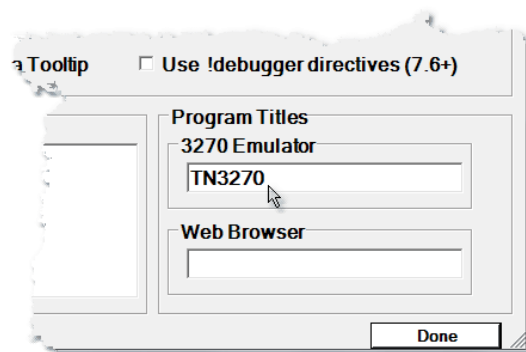
The **Break after READ SCREEN** setting that exists at the end of the Debugger Client session persists to the next run of the Client.

Note: As a convenience, the Debugger can bring your 3270 emulator application to the top on your PC screen when the Client pauses for the READ SCREEN or READ MENU.

To invoke this feature:

1. Select **Preferences** from the **File** menu (or use the Ctrl+P keyboard shortcut).
2. In the **Preferences** dialog box, locate the **3270 Emulator** text box (in the **Program Titles** section) and provide a text string that matches some or all of the title that displays at the top of the emulator window.

The characters in your matching string can be any case and match anywhere in the title. Any trailing blanks you enter are preserved.



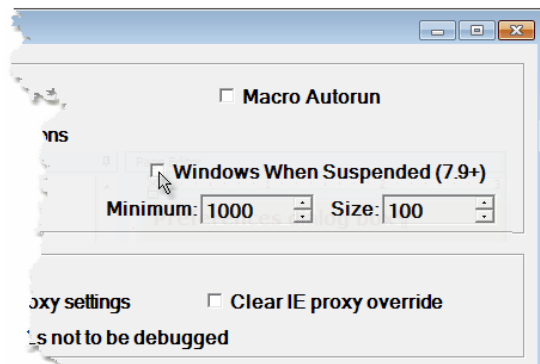
Note: If you are working with READ SCREENs or READ MENUs in a *Janus Web Legacy Support* application, your browser is the tool with which you respond. Therefore, to have the Debugger pop up your browser window when the Client pauses for the READ SCREEN or READ MENU, provide a browser-name-matching string in the **Web Browser** text box instead of in the **3270 Emulator** text box.

3. Click **Done**.

The feature takes effect at the next execution of a **READ SCREEN** or **READ MENU**. The **3270 Emulator** (or **Web Browser**) setting that exists at the end of the Client session persists to the next run of the Client.

Note: By default, this feature does not take effect if the Client is not debugging the part of a program that contains the **READ SCREEN** or **READ MENU**. For example, these statements might be in code selected to be [excluded from debugging](#).^[66]

However, if in addition to a **Program Titles** value you also select the **Windows When Suspended** option in the **Preferences** dialog box (**Execution Options** section), the feature will apply whenever these statements occur, even in code the Debugger is not actively executing. This will apply whether or not you have selected the **Break after READ SCREEN** checkbox.



3.1.5 Stepping out

If for any reason you no longer want to continue debugging or examining the subroutine, user-written *Janus Soap* method, or [daemon](#)^[141] that you are currently stepping through, you can discontinue debugging and leave ("step out" of) the subroutine, method, or daemon code and resume debugging on the statement following the statement that originally called the subroutine, method, or daemon.

This would be equivalent to having [set a breakpoint](#)^[58] on the statement after the subroutine, method, or daemon call, then clicked the **Run** button. You might also view it as a "pop" out of the level of code you are debugging and to the level of the calling code.

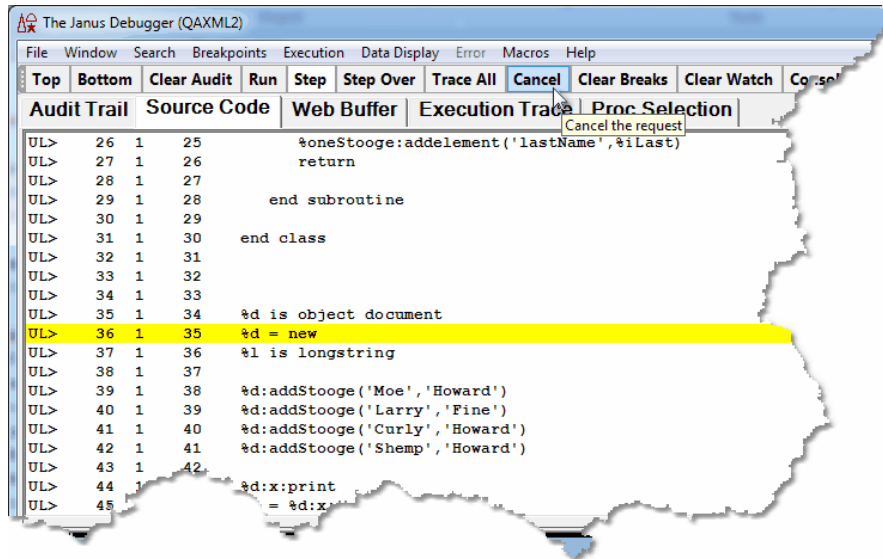
To execute a Step Out from within a simple or complex subroutine, a *Janus Soap ULI* user-written method, or a daemon, select **Step Out** from the **Execution** menu, or use [stepOut](#)^[270], the command equivalent.

If you step out from mainline code (that is, not a subroutine, method, or daemon), execution simply completes normally, as if you had clicked the **Run** button.

If you press the Enter key immediately after executing a step out, another step out is executed (as of Client Build 57) if program execution has not completed.

3.1.6 Cancelling execution

When you run a request that includes User Language, the Debugger Client displays the program in the **Source Code** tab, prior to the execution of any program statements. At this point or at any point after you begin to execute the code, you can terminate the execution of the program by clicking the **Cancel** control button.



When you click **Cancel**, you receive a "do you really want to" message because the operation is not reversible. You can view the remainder of the current program but cannot subsequently execute any of that code, unless you invoke it for debugging again.

If you click the **Cancel** button, you receive a **Do you really want to cancel?** prompt because the operation is not reversible. If you click Yes for the prompt:

- **Cancelled on request** is displayed in the status box.
- You can view the remainder of the current program but cannot execute any of it.
- The Client takes no further action until you click the **Run** button, which finishes the request (omitting the execution of the statements from the point at which you click **Cancel** until the request **End** statement). A cancellation message is then issued:

For a web request, the Debugger sends the browser a message like the following:



For a 3270 request, a message like the following is sent to the terminal:

***** MSIR.0943: Request cancel performed from debugger**

Note: The Ctrl+X key combination (by [default](#)^[291]) is equivalent to clicking the **Cancel** button.

3.1.7 Excluding sections of source code from debugging

Debugging a large application may involve the scanning of a large volume of code to get to the parts of the program that you need to debug. It can be cumbersome and time consuming to ship so much code to the Debugger Client and/or to inspect so many pages of code. To gain some time and space economy, the Debugger lets you debug some sections of your program while omitting others.

Two Debugger features let you exclude code from debugging:

- **Exclude/Include directives** let you mark blocks of code of any length to be excluded from display and debugging — but not from execution. You explicitly mark the beginning and ending of such blocks, or if they are procedures, *Janus* SOAP methods, or User Language subroutines, you specify them by name or by name pattern, and you can provide them by list. You can also similarly identify blocks of code within these excluded blocks that will **not** be excluded from display.
- **"Run Until" processing** operates on procedures only. You can have the Debugger run your program code without interruption until it reaches a procedure you want to display for debugging. You can identify this procedure by name or name pattern, and you can provide a list of such procedures in a "white list" or a "black list."

Run Until procedure processing differs from Exclude directive processing of procedures in that Run Until excludes complete programs, while Exclude directives (explicitly or implicitly) exclude parts of programs (those called by "inner procedures"). An inner procedure is within a Begin/End block, invoked by a User Language **INCLUDE statement**. A procedure invoked by a command-level **INCLUDE** is an "outer" procedure. Once a procedure is running in the Debugger and a request Begin is seen, Run Until procedure detection does not stop until after the End of the request. It ignores inner procedures and looks only for outer procedures.

The **Proc Selection** page in the Client contains most of the controls for both types of code exclusion techniques.

Exclude and Include directives are described in:

[Selectively excluding source code blocks](#)^[67]

Run Until processing may be invoked once for a single specified procedure, or it may be applied to any of multiple procedures specified in a list. Run Until processing is described in:

[Running to a specific procedure](#)^[75]

[Running only to listed procedures](#)^[79]

3.1.7.1 Selectively excluding source code blocks

Sirius User Language Macro Facility statements called **Debugger directives** let you exclude one or more blocks of source code or entire procedures, methods, or User Language "complex" subroutines from interactive debugging.

"Excluding" code from interactive debugging means:

- The excluded code is *not* displayed in the Debugger **Source Code** tab, nor are the excluded source code lines sent from the mainframe to the client.
- You may *not* step through or set breakpoints in the excluded code.
- Excluded statements are *not* shown in the results of Debugger execution tracing or statement history displays.
- However, the excluded code *is* executed normally (there is no difference in the runtime evaluation of the request), and the *Janus Debugger Audit Trail* and **Web Buffer** tabs will show output from excluded statements.

This feature is useful both for tidying your source code display (removing non-pertinent sections of code) and for decreasing the download time of source code sent from the mainframe to the Client.

The Debugger directives have the following format:

!debugger *directive*

Where *directive* may be one of the following:

```
exclude on
exclude off
exclude proc pname_or_pattern
exclude routine rname_or_pattern
```

```
include on
include off
include proc pname_or_pattern
include routine rname_or_pattern
```

Within excluded code you can specify blocks of lines that will **not** be excluded and will be presented for debugging; for example, an important subroutine or method. You indicate such non-excluded blocks by **Include** directives.

As described in [Using the code-exclude feature in its normal mode](#)^[68], Debugger Exclude and Include directives are also implied and invoked if you use the buttons in the **Exclude Parts of Programs from Debugging** section on the **Proc Selection** page to specify lists of procedures or lists of subroutines or methods to exclude/include.

As described in [Using Init Exclude mode](#)^[74], another approach to debugging a small block of code within a larger excluded block is to invert the way the code-exclude feature operates: Instead of including all code from the beginning of the request until an Exclude directive, the "Init Exclude" variation of the feature initially excludes all code from the beginning until an Include directive. You invoke Init Exclude mode from a Client menu item or command (or mapped button or hot key).

User Language Macro Facility statements, which start with an exclamation character (!), are described in the Sirius wiki at <http://sirius-software.com/index.php/UserLanguageMacroFacility>.

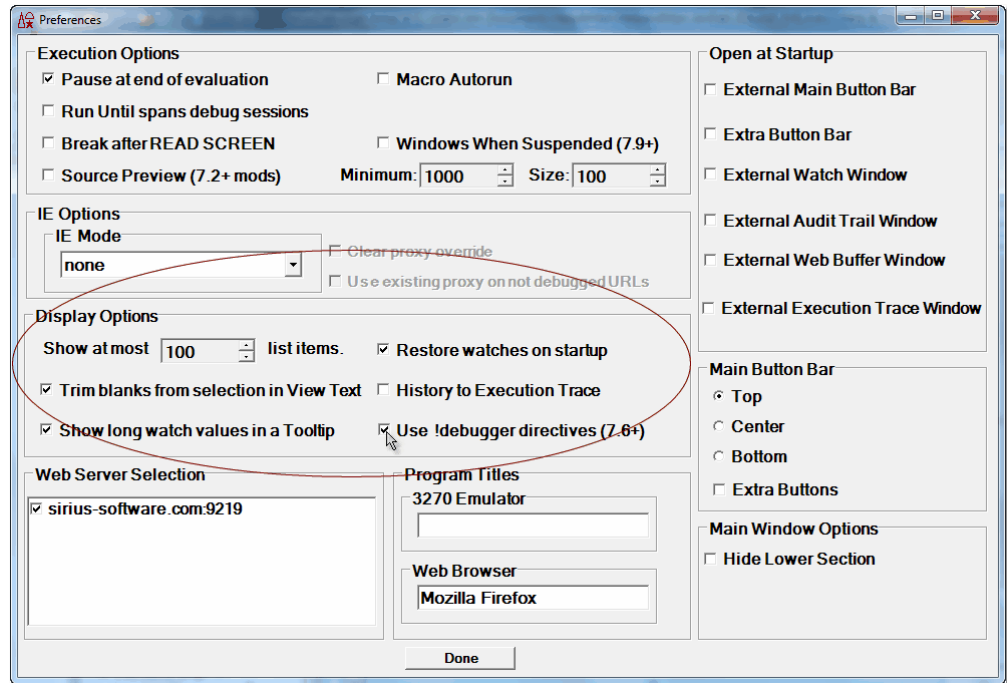
Using the code-exclude feature in its normal mode

In any single compilation unit (BEGIN statement to END), you may exclude as many as 300 blocks of code from debugging, and within excluded blocks, you may designate as many as 300 blocks of code that will not be excluded. (The block limits are 40 if *Sirius Mods* version is 7.8; 20 if prior to 7.8.) A block may be a designated group of code lines or a named routine or procedure.

To exclude code:

1. Enable Debugger directives. By default, the feature is not enabled (no code is excluded from debugging).
 - a. From the Client **File** menu, select the **Preferences** option.

- b. In the **Display Options** section of the **Preferences** dialog, select the **Use !debugger directives** option:



2. In your source code, specify an **Exclude** directive to indicate the code to be excluded from debugging:

- To exclude any block of consecutive lines, indicate the beginning of the block to be excluded:

!debugger exclude on

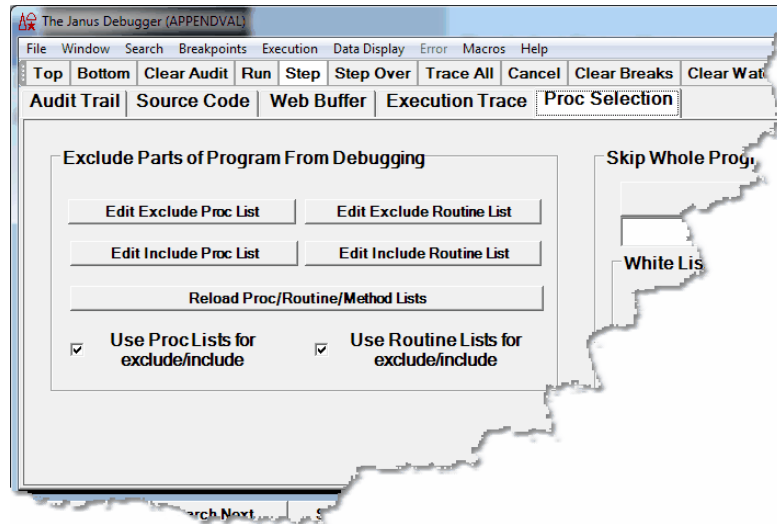
This statement must appear between a **BEGIN** and **END** statement — it may not be used at command level.

- To exclude a particular procedure, or the code that defines a method or subroutine:
 - Use the appropriate keyword and case-insensitive name in the directive. You can also use wildcards (described [below](#)⁷⁰) to form a name pattern. Also, for a method or User Language subroutine, use **routine**:

```
!debugger exclude proc MyProc*
!debugger exclude routine MyMethod
```

Or:

- Use the **Edit Include Proc List** or **Edit Include Routine List** buttons on the **Proc Selection** page to provide a list of the procedures or a list of the routines you want to exclude:



These buttons open a blank `excludeProc.txt` or `excludeRoutine.txt` file in Microsoft's *Notepad* or in your [local editor](#)^[166]. In the files, you specify the items you want to exclude, observing the following syntax rules:

- One entry per line
- Leading and trailing blanks are ignored
- Case matching is insensitive
- Wildcard matching using asterisk (*), question mark (?), and double quote (") is allowed (see [Shortcuts for procedure names](#)^[76])
- Blank lines are ignored
- Any line starting with a number sign (#) is treated as a comment

After you save and Exit the file(s), click the **Reload Proc/Routine/Method Lists** button to make the file content known to the Client. Equivalent to using this button is the [reloadLists](#)^[239] Client command.

The **Use Proc Lists for exclude/include** and **Use Routine Lists for exclude/include** checkboxes enable and suspend the feature. They are selected by default. For earlier builds, only the **Use !debugger directives** option on the **Preferences** dialog box controls the feature. If you are running under a *Sirius Mods* version lower than 7.9, you must select one or both of the **Proc Selection** page checkboxes as well as the **Use !debugger directives** option; under 7.9 or higher, the **Proc Selection** page checkboxes alone are sufficient to control the feature.

Note: Whether or not you select these checkboxes does **not** affect Debugger recognition of any `!debugger` directives you specify in source code outside of these lists. Such directives are controlled only by the **Use !debugger directives** option.

Your checkbox selections are remembered in subsequent Client sessions.

3. If excluding a block of lines (that is, not a named procedure or routine), indicate the last line to exclude.

From the **exclude on** directive, source code lines are excluded until the first of the following is encountered:

- A **!debugger exclude off** statement
- The end of the compilation (END statement is reached)
- The end of the procedure that contains the starting **exclude on** directive
- An Include directive (described next)

4. Indicate (with Include blocks) any lines *within the excluded code* that you do **not** want to be excluded:

If a (non-named) block of lines:

- a. Specify an **include on** directive to indicate the beginning of a section of code you want to be included in debugging:

!debugger include on

- b. Indicate the last line to include.

From the **include on** directive, source code lines are included for debugging until the first of the following is encountered:

- A **!debugger include off** statement
- The end of the Exclude block that contains the starting **include on** directive
- The end of the compilation (END statement is reached)
- The end of the procedure that contains the starting **include on** directive

If a named procedure or routine:

- Specify after the Exclude directive and before the procedure or routine an **include proc name_or_pattern** or **include routine name_or_pattern** directive to indicate the code you want to be included in debugging.

Or:

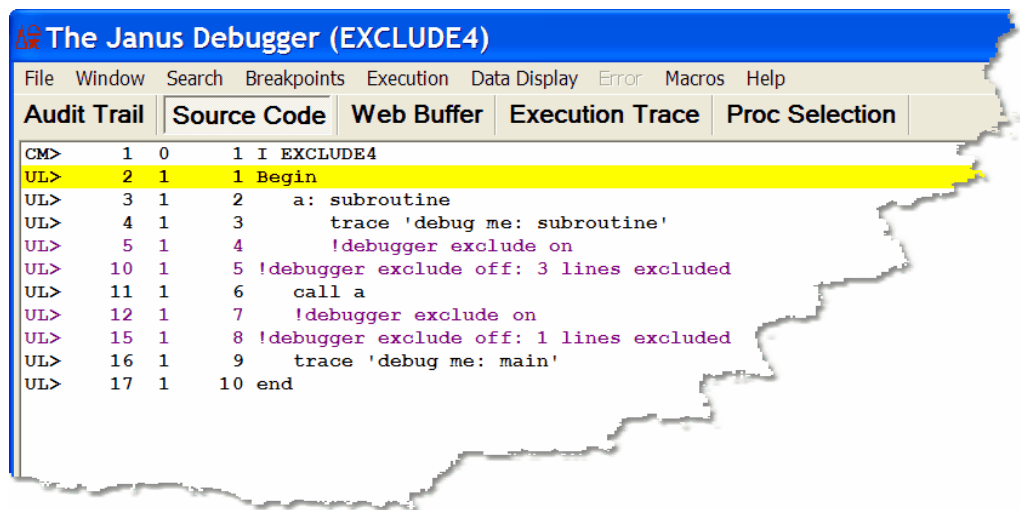
- Specify the procedures or routines in a list file you access via the **Edit Include Proc List** or **Edit Include Routine List** buttons on the **Proc Selection** page (see image and description above).

In the file, specify the items you want to include (that are located within code you are excluding).

After you save and Exit the files, click the **Reload Proc/Routine/Method Lists** button (or use the `reloadLists` command) to make the file content known to the Client.

- Invoke and debug the code.

In place of the excluded lines, the Debugger inserts a comment to the `exclude off` directive in the Client **Source Code** display that specifies the number of lines excluded. Any lines within the excluded block that contain Include directives are also denoted. All directive statements are shown in purple text [by default](#).^[293]



Usage notes:

- If you define more than 300 Exclude blocks, or more than 300 Include blocks, an error is issued. (These block limits are 40 if *Sirius Mods* version is 7.8; 20 if prior to 7.8.) A block may be a designated section of consecutive code lines or a named routine or procedure.
- Until an `exclude on` directive ends, subsequent `exclude on` statements are ignored. This is also true for the implied `exclude on` created by Init Exclude mode (described in the subsection below).

- A `!debugger include on` directive has an effect only within an Exclude block. For example:

```
!debugger exclude on
... lots of code that is not debugged
!debugger include on
... an important subroutine that needs debugging
!debugger include off
... lots more code that is not debugged
!debugger exclude off
```

- Dummy string substitution is done on `!debugger` directives **before** they are parsed and processed. This lets you build a directive conditionally, perhaps to keep code compatible with *Sirius Mods* versions earlier than 7.6.

For example:

```
begin
  trace $sirver
  if ($sirver >= 706) then
    $setg('COMMENT','')
  else
    $setg('COMMENT','*')
  end if
end
begin
  * Code to debug
  trace 'Debug me'
  ?&COMMENT !debugger exclude on
  * Code not to debug
  trace 'Do not debug me'
  ?&COMMENT !debugger exclude off
end
```

- When you compile a program that contains Exclude blocks or Exclude and Include blocks, *Model 204* writes a summary of the exclusions to the Audit Trail at the end of compilation:

```
MS MSIR.1003: 2 Debugger Exclude block(s) defined
MS MSIR.1004: Lines 5-9 excluded from the debugger
MS MSIR.1005: Quads 24-135 excluded from the debugger
MS MSIR.1004: Lines 15-18 excluded from the debugger
MS MSIR.1005: Quads 160-271 excluded from the debugger
```

- The Client [setPreference](#)^[260] command has options that let you toggle the **Use ! debugger directives**, **Use Proc Lists for exclude/include**, and **Use Routine Lists for exclude/include** checkboxes.

Using Init Exclude mode

By default, starting from your program's BEGIN statement, the code-exclude feature preserves for your debugging all the code lines that you do not explicitly exclude via directives. However, you may have cases where it would be advantageous to invert the default behavior, that is, starting from the BEGIN statement, to exclude all the code from debugging except the blocks you explicitly preserve via directives. "Init Exclude" mode provides such an inversion of the default.

Init Exclude mode is the equivalent of explicitly specifying **!debugger exclude on** immediately following your program's BEGIN statement. It changes nothing else about the operation of the code-exclude feature. All program code lines are excluded from debugging until one of the following directives is encountered explicitly or is encountered implicitly via specification of an Include Proc or Include Routine list (**Proc Selection** tab):

```
!debugger include on
!debugger include proc
!debugger include routine
!debugger exclude off
```

Subsequent code is then included until the first of one of the following:

- A **!debugger include off** or a **!debugger exclude on** statement
- A **!debugger exclude proc** or a **!debugger exclude routine** statement (explicit, or implicit via **Proc Selection** page button)
- The end of the compilation (END statement is reached)
- The end of the procedure that contains the starting **include on** directive

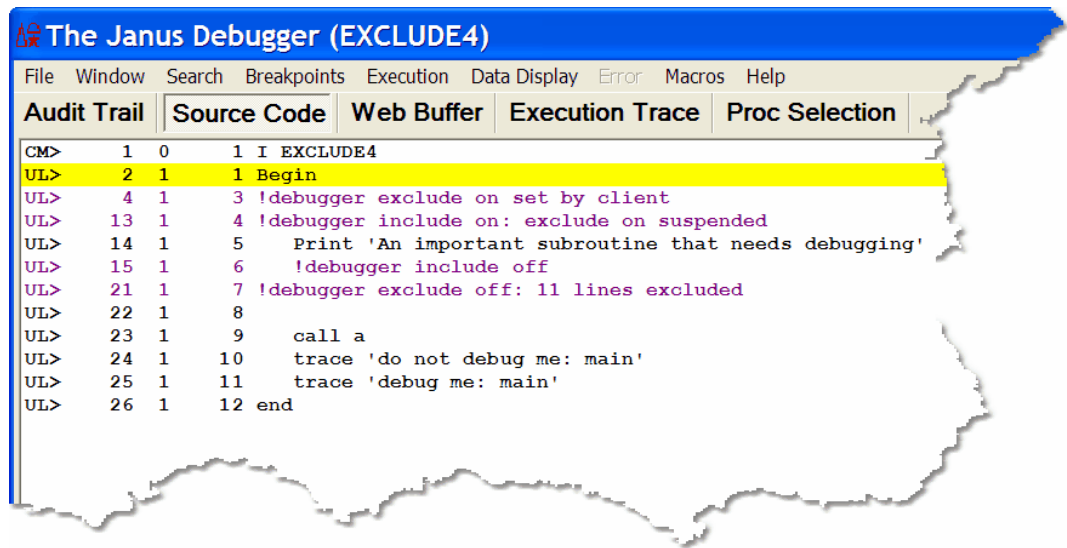
To invoke Init Exclude mode:

1. Enable Debugger directives by selecting the **Use ldebugger directives** option from the Client **File** menu, **Preferences** option.
2. Do either of the following:
 - Select **Toggle Init Exclude** from the Client's **Execution** menu.
 - Select the Client button, hot key, or macro you [configured](#)^[285] to execute the [toggleInitExclude](#)^[273] command.

An **Init Exclude mode is on** message in the Client's [Status bar](#)^[50] as well as a checkmark next to the **Toggle Init Exclude** option in the **Execution** menu confirm that Init Exclude mode is on.

3. Run the program you want to debug.

A "!debugger exclude on set by client" comment in the code indicates that lines were excluded from the beginning of the program:



To leave Init Exclude mode, you repeat step 2, above.

See Also

[Running to a specific procedure](#) ⁷⁵

[Running only to listed procedures](#) ⁷⁹

3.1.7.2 Running to a specific procedure

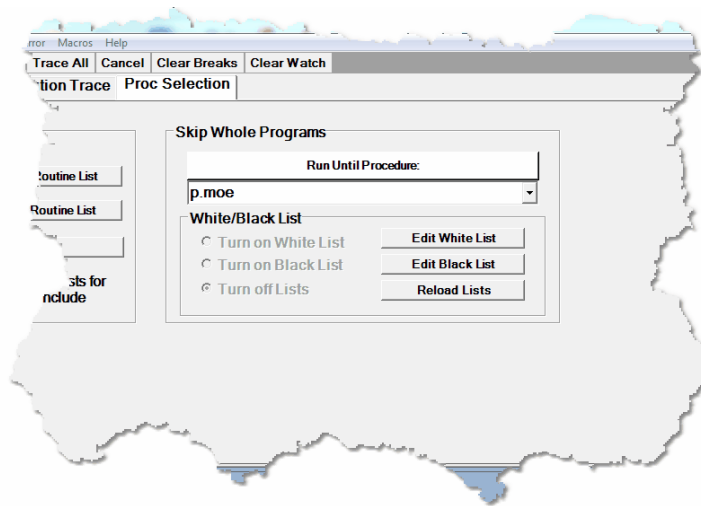
You can direct the Debugger to run an application's code without interruption until it reaches a specific outer, command-level, procedure, and then to display that procedure for debugging.

For example, you are having a problem with the procedure P.MOE, but the normal flow of the application moves through procedures LARRY, SEMP, and CURLY prior to P.MOE. The three preceding procedures are known to work fine, and you don't want to interactively debug them.

To run the application normally, executing the preceding procedures, but stopping at P.MOE to begin debugging:

1. Beginning with your application program displayed in the **Source Code** page, select the **Proc Selection** tab.

2. In the text area below the **Run Until Procedure** button in the **Skip Whole Programs** area, specify the name of the procedure at which to begin debugging (this input is *not* case sensitive).



3. Click **Run Until Procedure**.

The application executes until it reaches the specified procedure, then the **Source Code** page highlights the first executable User Language statement in that procedure. You are ready to debug.

You can also execute Run Until processing by using:

- The **Execution** menu **Run Until Proc** option, which will run immediately, using the procedure name currently specified on the **Proc Selection** tab
- The [mappable](#)^[285] Client command [runUntil](#)^[245]
- A white list, as described in [Running only to listed procedures](#).^[79] Run Until for a specific procedure is the same as a white list that contains a single procedure.

Shortcuts for specifying procedure names

Instead of specifying the whole procedure name in the **Run Until Procedure** text box:

- You may be able to find the name you want by clicking the arrow to the right of the **Run Until Procedure** text box.

This reveals a drop-down list of the names of as many as the last twenty procedures you entered (for this and from previous sessions). These names are also saved in the `until.txt` file.

- You can also use leading, trailing, or intermediate wildcards to form a name pattern to stop the Debugger on the first procedure that matches the pattern:
 - An asterisk (*) represents any string of characters.
`*.M*` would stop the code execution at procedure `ACC.MOE`, or `HOME.MARY`, or `P.MOO`, and so on.
 - A question mark (?) represents any single character.
`?.*` matches `P.MOO`, but not `ACC.MOE`.
 - A double quote (") escapes wildcard translation of the asterisk or question mark that follows it.
`?."*` matches `P.*`, but not `P.MOO`.

Prior to *Sirius Mods* version 7.9, only an asterisk wildcard is allowed.

Note: The *Model 204* LAUDPROC (Length of Audit Procedure Names) User 0 parameter (default: 21) must be set to the size of the largest procedure name that will be filtered. Otherwise, name matching is done against truncated procedure names.

Precedence and scope for Run Until

As described for the [Break method](#)^[162], the "run until" processing takes precedence over any Break method calls that may be present in your procedures. Run Until ignores such calls.

Run Until also continues without interruption past any persistent-session suspend/resume sequences.

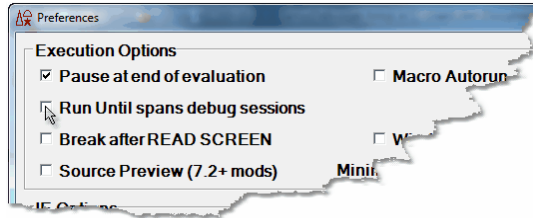
The default boundaries beyond which Run Until does *not* continue processing are these:

- The end of the debugging session
 Once a session ends, either normally or through a lost connection, and Run Until searching is interrupted before it finds a specified procedure, a restarted Client does not automatically resume its search.
- The end of an HTTP request (*Janus Debugger*)
 If the current HTTP transaction completes and the Debugger does not encounter the target procedure, it does not continue its search for the target into subsequent HTTP requests. This is ordinarily not an issue.

It may be the case, however, that you want the Debugger *not* to respect these boundaries. You do not want to have to repeatedly invoke Run Until for the same procedure. For example, you are working with HTML frames, where each frame and frameset is a separate HTTP transmission to the browser, and you want Run Until to span all these HTTP requests rather than stopping for each frame that involves User Language.

To change the span of Run Until:

1. In the Client, select **Preferences** from the **File** menu (or use the Ctrl+P keyboard shortcut).
2. In the **Preferences** dialog box, select the **Run Until spans debug sessions** checkbox (which is clear by default), then click **Done**.



Now, with a single invocation of Run Until, you can be sure to display the procedure you want. If, for example, the *Sirius Debugger* was turned off and then on again, the Client will resume an incompleted Run Until search. Or in the case where you are debugging a request that satisfies an individual frame or frameset, the Client will continue (with no other execution breaks) through the code for subsequent frames until it finds the specified procedure.

This setting takes effect immediately, and it persists over multiple runs of the Client.

Interrupting Run Until processing

Run Until processing is manually interruptable: the mappable Client command **breakOnNextProc** lets you override a Run Until to interactively debug the next included procedure.

To use a manual interrupt for Run Until processing:

1. Map the **breakOnNextProc** command to a Client button or hot key (as described in [Setting up the ui.xml file](#)^[287] or in a macro (as described in [Using Debugger macros](#)^[309]).
2. While debugging an application where Run Until processing is active, invoke the button or hot key or macro.

Break on next proc set displays in the Client's **Status bar**^[50] to indicate the command has been successfully issued, and the next procedure included at command level (or from APSY) will be debugged, even if the Run Until would normally ignore it.

You cannot invoke an interrupt while debugging sdaemon code.

In the same way you interrupt Run Until processing, you can also interrupt [White List procedure processing](#).^[82]

See Also[Selectively excluding source code blocks](#)^[67][Locating and editing procedure source files](#)^[126]**3.1.7.3 Running only to listed procedures**

From a large domain, you may only want to interactively debug a particular set of procedures, say the procedures for which you are responsible. [“Run Until Procedure” processing](#)^[75] lets the Debugger run through code until it reaches a procedure whose name matches a name or pattern you specified. But it may be simpler to forego the repeated specifying of individual names and instead to debug just the procedures pre-specified in a list. In the Debugger, this is "White List" processing. Alternatively, it might be easier to pre-specify in a "Black List" the procedures you do *not* want to debug.

To enable White or Black List processing, you:

1. Provide a file that contains either the names of the procedures to be debugged or a file that contains the names of the procedures *not* to be debugged
2. Invoke the feature in the Client GUI

Black List processing is new in Client Build 62.

Enabling and updating white or black list processing is discussed further in the following subsections:

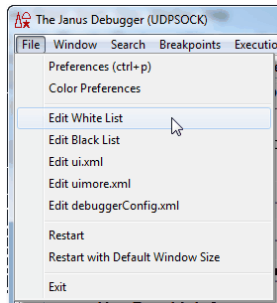
[Setting up a White or Black List file](#)^[79][Invoking White or Black List processing](#)^[81][Updating a White or Black List file](#)^[82]**Setting up a White or Black List file**

You must create a simple ASCII text file named `whitelist.txt` or `blacklist.txt`. In `whitelist.txt`, you list the *Model 204* procedures that you want to debug; in `blacklist.txt`, you list those you do not want to debug. The Debugger Client will create these files for you (using Microsoft's *Notepad*, as described below). Or, you can create "manually" the file you want, in the same folder as the Debugger Client executable file (JanusDebugger.exe), using any text editor.

If you provide no `whitelist.txt` or `blacklist.txt` file, the user interface for White List or Black List invocation is disabled.

In the Debugger Client:

1. From the **File** menu, select **Edit White List** or **Edit Black List** (or, as mentioned below, use the **Edit White List** or the **Edit Black List** button in the **White/Black List** box in the **Proc Selection** page).



A blank *Notepad* **whitelist.txt** or **blacklist.txt** file is opened for you. (If such a file already exists, that file is opened.)



2. If a **whitelist.txt** file, specify the procedures you want to debug; if a **blacklist.txt**, specify those you don't want to be debugged.

Note the following syntax rules:

- One entry per line
- Leading and trailing blanks are ignored
- Case matching is insensitive
- [Wildcards](#) are allowed
- Blank lines are ignored
- Any line starting with a number sign (#) is treated as a comment

Here is an example of a valid file specification:

```
# white list the three stooges
p.moe
p.larry
p.shemp
```

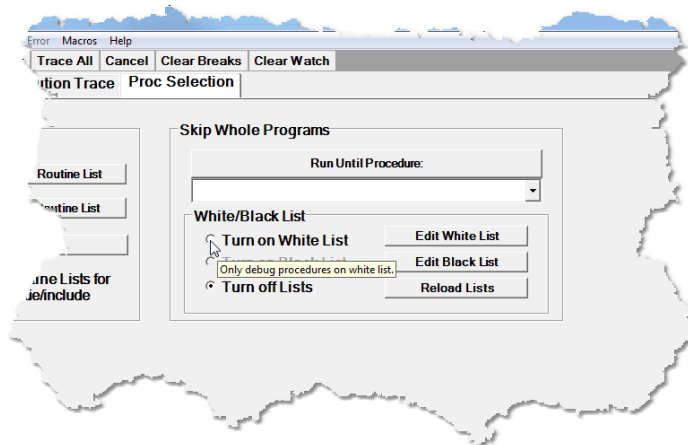
3. Save and Exit the file.

Invoking White List or Black List processing

If the Debugger Client detects a valid White List or Black List file, it enables the option to turn the appropriate feature on or off.

To invoke white or black list filtering at any time (it is off by default):

1. Select the Client's **Proc Selection** tab.
2. If the Client was already started before you created or updated the `whitelist.txt` or the `blacklist.txt` file, click the **Reload White List** or **Reload Black List** button so the Client can detect the file. This enables the button that invokes White List or Black List processing, and it also reports the file's contents to the Client log file (`log.txt`).
3. In the **White/Black List** box in the **Skip Whole Programs** area, click the **Turn on White List** or **Turn on Black List** button.



Once you click the **Turn on White List** button, for example, `White list is active` displays in the [Status bar](#)^[50], and white list filtering is enabled for the session until you turn it off by clicking the **Turn off Lists** button. As of Client Build 62, your filtering selection persists over runs of the Client until you undo it.

The Client's **Execution** menu also has **Turn On White List** and **Turn Off White List** and **Turn On Black List** and **Turn Off Black List** options that have the same effect as the **Proc Selection** tab buttons.

When white or black listing is on, the Debugger filters outer procedures automatically, stopping to interactively debug only the requests that are on the white list or not on the black list. A procedure not on the white list or on the black list still executes normally, but it is not interactively debugged, and the Client's **Audit Trail** displays are immediately refreshed to specify that such a procedure has been "skipped."

You can also enable and disable White List or Black List processing by using the [mappable Client commands](#)^[285] that are equivalent to the above steps. The commands are `turnOnWhiteList`, `turnOnBlackList`, `turnOffWhiteList`, and `turnOffBlackList`.

Interrupting White List processing

Once White List or Black List processing is invoked, it continues in effect unless you explicitly turn it off or until you exit the Client. It is manually interruptable, however: the mappable Client command `breakOnNextProc` lets you override the White List or Black List to interactively debug the next procedure.

To use a manual interrupt for White List or Black List processing:

1. Map the `breakOnNextProc` command to a Client button or hot key (as described in [Setting up the ui.xml file](#))^[287] or in a macro (as described in [Using Debugger macros](#)^[309]).
2. While debugging an application where White List or Black List processing is active, invoke the button or hot key or macro.

`Break on next proc set` displays in the Client's [Status bar](#)^[50] to indicate the command has been successfully issued, and the next procedure included at command level (or from APSY) will be debugged, even if the active White List or Black List would normally ignore it. When this next procedure is debugged, the **Audit Trail** will display, for example, "White list accepted: *procedurename*."

The interrupt applies only to a single procedure, and you must manually invoke it again for each subsequent procedure you want to exclude from White List or Black List processing.

You cannot invoke an interrupt while debugging sdaemon code.

In the same way you interrupt White List or Black List processing, you can also interrupt [Run Until procedure processing](#).^[78]

Updating a White List or Black List file

To update a White List, for example, at any time, from the Debugger Client:

1. Select either of the following two options (which let you update and save in *Notepad* the current `whitelist.txt` file)
 - The **Edit White List** button in the **White List** box in the **Proc Selection** page
 - The **Edit White List** option of the **File** menu
2. On the **Proc Selection** page, click the **Reload White List** button (or execute the [reloadWhiteList](#)^[238] command).

After a successful reload, `White list reloaded` displays in the [Status bar](#)^[50].

Alternatively, you can use your text editor to modify a White List or Black List file at any time, after which you click the **Reload White List** or the **Reload Black List** button on the Debugger Client **Proc Selection** page.

See Also

[Selectively excluding source code blocks](#)^[67]

[Locating and editing procedure source files](#)^[126]

3.1.8 Altering the flow of execution

When debugging, you might want to test a code fix by resetting a variable's value and then re-executing one or more statements. Or you might want to alter the flow of control in the program to ensure that hard-to-reach code is tested, such as error paths. You can accomplish these tasks using the Debugger's Jump feature.

The Jump feature lets you transfer control to a statement and then execute that statement. The target statement may be earlier or later in the request than the current statement.

You invoke a jump by right-clicking a line in the **Source Code** window or by [using commands](#)^[84]. The commands offer additional functionality: jumps to a line number, jumps that are a number of lines relative to the current line, and jumps to lines that contain specified strings.

Manually executing a jump

To perform a jump from a **Source Code** line to a statement you manually select:

1. Right-click the target line to which you want to transfer program execution.
If the target is the first line of an executable statement, the context menu will contain a **Jump Here** option.

2. Select **Jump Here**.

The Debugger validates this target line, using the rules described below in [Jump validation rules](#)^[84].

If the target is valid:

- a. Control is immediately transferred to the target statement (intervening statements are not executed).
- b. The target statement executes.

- c. The next executable statement after the target is highlighted, and program execution pauses.

If the target is not valid: control is *not* transferred, and an **Invalid Jump** message is displayed in the [status bar](#)^[50].

Note: Be aware that the execution of a statement you validly jump to may result in an unexpected request error because the jump bypassed the execution of statements that were logically-necessary predecessors to the target statement.

Invoking a jump from a macro or mapped command

The [jumpToLine](#)^[216] and [jumpToMatch](#)^[217] commands let you add a jump to a macro or Client button or hot key. With these commands, since you cannot manually select the line to which to jump, you select the target lines by number or by matching a string.

Jumping to a target line by absolute or relative line number

Like a manually-executed jump, the **jumpToLine** command transfers control to a specified request statement within the **Source Code** page, then executes that statement. You identify the target statement by supplying a keyword or a number:

jumpToLine [**current** | *number*]

For further information about specifying the command, see [jumpToLine](#).^[216]

Jumping to a target line by matching a string

The **jumpToMatch** command transfers control to a request statement within the **Source Code** tab that contains a specified matching string, then it executes that statement. The target statement is the first statement *from the top* (first) line in the **Source Code** page that contains a match for the string you provide.

For further information about specifying the command, see [jumpToMatch](#).^[217]

Jump validation rules

A jump operation is allowed if it follows these rules:

- You must have executed at least one statement in the request.
- Jumps are confined to the current nesting level. You may jump within but not into or out of nested code, which includes the following User Language constructs:
 - Loops (For, Repeat)
 - Subroutines (both simple and complex)
 - Janus SOAP methods
 - On units (all types)

For example, once the Client's current-line indicator moves from the statement that calls a user-defined method into the code that defines the method, any attempted jump to a statement outside the method definition is invalid. And any attempted jump into the definition code from the method-calling statement (or from any statement outside the method definition) is invalid.

Note: Although you may not jump out of nested code, you can [step out](#)^[64] of the current level and resume debugging on the statement following the original call of the nested code.

- The `jumpToLine` and `jumpToMatch` commands transfer control only to target statements that are executable.
- The `jumpToLine` command indicates the target statement absolutely by [statement line number](#)^[11] or relatively by a number of lines forward or backward from the current line. It is an invalid jump if you absolutely specify the statement line number of an empty line, but it is valid to jump relatively to an empty line (in which case, the jump attempts to go to the line following the empty line).

3.1.9 Previewing program code

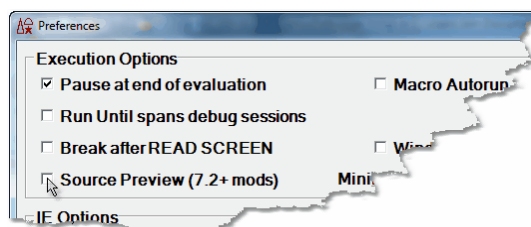
If you work with User Language programs that require an appreciably long time to download for debugging, due to their size alone or to a slow network connection or high network traffic, you may prefer to use a Debugger feature that lets you inspect a small "chunk" of the beginning of the program to decide whether to download the entire program for debugging or just to run it.

The Source Preview feature lets you download a program's initial 100 lines (at minimum). After inspection of this code chunk, you can download the entire program as usual or you can simply run the program without downloading it.

Enabling Source Preview

To enable the Source Preview feature (which is disabled by default):

1. In the Client, select **Preferences** from the **File** menu (or use the Ctrl+P keyboard shortcut).
2. In the **Preferences** dialog box, in the **Execution Options** section, select the **Source Preview** checkbox (it is clear by default).



3. The **Minimum** value (to the right of the **Source Preview** checkbox) is the default for the minimum length a source program must be in order to be previewed. No programs less than 1000 lines may be previewed. Modify this setting (increments of 1000; maximum of 100,000) if you want previewing available only for longer programs.
4. The **Size** value (to the right of **Minimum**) is the default for the size of the preview code chunks. Modify this setting (increments of 100; maximum of 10,000) if you want larger chunks.
5. Click the **Done** button to save your settings and enable the feature.

Operating in preview mode

When Source Preview is enabled and you encounter a program with a number of lines greater than or equal to the Source Preview **Minimum** setting, the following happens:

1. As many as Source Preview **Size** lines of the program are sent to the Client **Source Code** page.
2. The [Status bar](#)^[50] confirms that you are in preview mode and informs whether the source program compiles. It displays either **Preview: good compile** or **Preview: bad compile**. Even if the program does not compile, you will still have the option to download and view the entire program and its embedded compilation error messages.
3. You inspect the preview and decide whether to download the entire program for debugging or to skip the debugging of the program. Or, if the program did not compile, you decide whether to download and view the entire program or to skip it.
 - To download the entire program for debugging:

Select the **Debug Previewed Source** option from the **Execution** menu, or use a button or hot key you mapped to the [skipPreview](#)^[268] command, or click the Client's **Step** button or perform a **Source Code** text search.
 - To download the entire program to view compilation error messages:

Select the **Debug Previewed Source** option from the **Execution** menu, or use a button or hot key you mapped to the [debugPreview](#)^[201] command.
 - To skip the program:

Click the Client's **Run** button, or select the **Skip Previewed Source** option from the **Execution** menu, or use a button or hot key you mapped to the **skipPreview** command.

3.2 Viewing and modifying program elements

The Debugger Client provides two principal ways of examining the values of individual items in your source code:

- The **Watch Window** shows you the current values of as many items as you add to the window. If these values are changed by your code, their updated values are shown in the **Watch Window** the next time the Debugger execution pauses.
- The quick-display **Value** window shows you the current value of the single item you are selecting by a right-click on the line of code that contains it. Alternatively, the Client **Value** button displays in a Value window the value of the item you explicitly enter in the text box above the **Watch Window**.

You can use either of these approaches for most of the code elements whose values are viewable.

These sections are included:

[Watching program data items](#)^[87]

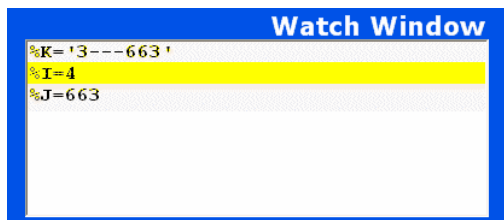
[Displaying temporarily the value of a program data item](#)^[101]

[Setting the value of a variable](#)^[124]

3.2.1 Watching program data items

The **Watch Window** box can display the current value of one or more program code data items. It is located in the lower right corner of the main window, unless it is [opened in a separate window](#)^[300] external to the Client.

The **Watch Window** is updated as the program runs, each time execution is paused by the Debugger. If the value of a watched item is changed by the last statement execution, it is highlighted:



The following types of items can be “watched”:

- %variable scalar values (for example: %x, %y)

- Elements of %variable arrays (for example: %names(%I), %names(34))
- Image items (including image array elements) (for example: %input:name)
- Global variables
- Database fields
- *Model 204* parameters
- Elements of \$lists and Stringlist and Arraylist objects
- Counts of items in a \$list or Stringlist or Arraylist
- Certain \$function calls: \$STATUS and \$STATUSD, \$CURREC, \$FIELDGROUPID, and \$FIELDGROUPOCCURRENCE
- *Janus SOAP* structure elements (for example: %address:city)
- *Janus SOAP* object variables (for example: %xmlInput:serial)
- *Janus SOAP* user-defined class member variables

Note: If any of the above items has subscripts or parameters within parentheses (for example, a %variable or \$list array item or a parameter in a *Janus SOAP* class method) that are *not* simple variables or constants, the item *cannot* be watched in the Debugger.

These subsections follow:

[Adding and removing Watch Window items](#)^[88]

[Saving and restoring Watch Window contents](#)^[91]

[Getting a detailed view of the value of a watched item](#)^[93]

[Watching Model 204 fields](#)^[95]

[Watching global variables](#)^[96]

[Watching object variables](#)^[97]

[Watching \\$lists and Stringlists](#)^[98]

[Watching class member variables](#)^[100]

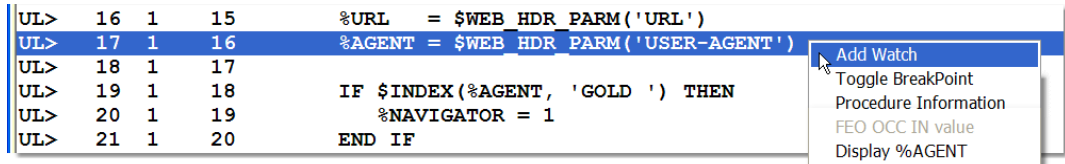
3.2.1.1 Adding and removing Watch Window items

You explicitly add to and remove from the **Watch Window** the items whose values you want to view.

Displaying items

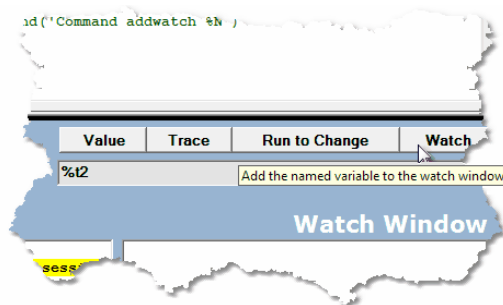
These are the simplest ways to add an item to the **Watch Window**:

- In the **Source Code** display, right-click a program line that contains variables, and select **Add Watch** from the context menu. All variables on that line are added to the **Watch Window**.



The [mappable](#)^[285] Client command that performs the same action is [addWatchOnCurrentLine](#).^[180]

- For the program line in the **Source Code** display that is highlighted to indicate it is *in the current execution position*, an alternative to right-clicking the current line is to select **Add Watch on Current Line** from the Client's **Data Display** menu. All variables on the line are added to the **Watch Window**.
- Type the name (case is not important) of the item in the [Entity-name input box](#)^[51], then click the **Watch** button (or select the **Data Display > Add Watch** menu item):



The mappable Client command that performs the same action is [addWatch](#).^[180]

- For these types of items, you must specify more than just the item name in the text box:

[Model 204 fields](#)^[95]

[User Language global variables](#)^[96]

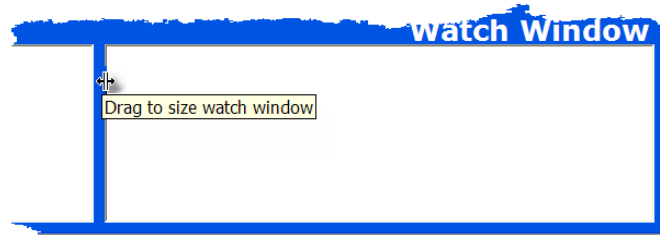
[Janus SOAP shared object variables](#)^[97]

[Sirius \\$lists](#)^[98]

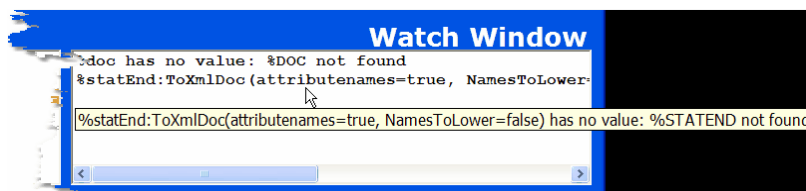
Viewing long items

To display lengthy values, you can do one of the following:

- Expand the **Watch Window** by dragging its left edge to the left:



- Hover your mouse over the item to display it entirely in a tooltip box:

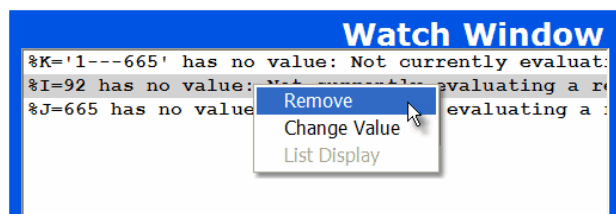


This feature is enabled by selecting the **Show long watch values in a Tooltip** option in the Client's [Preferences box](#)^[18] (it is off by default).

- Double-click the value to [display it in a Value window](#)^[93].
- Display the entire **Watch Window** in a [window that is external to the Client](#).^[300]

Removing items

To remove a single item from the **Watch Window**, right-click the item and select **Remove** from the context menu:



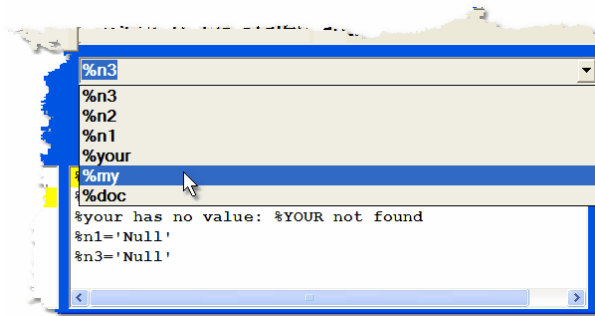
The mappable Client command that performs the same action is [removeCurrentWatch](#).^[240]

To remove all items from the **Watch Window**, click the **Clear Watch** button, below the main window. This button empties the **Watch Window** and instructs the mainframe portion of the Debugger to stop collecting any watch data.

The mappable Client command that removes all **Watch Window** items is [clearWatch.](#)^[179]

Redisplaying items

Each item you add to the **Watch Window** is also added to a drop-down list you can view in the [Entity-name input box](#)^[51] by clicking the arrow at the right. To redisplay an item you previously removed from the **Watch Window**, simply select from the drop-down list the item you want to watch again:



The items in the drop-down list persist across Debugger Client sessions. The contents of the **Watch Window** itself are redisplayed in subsequent Client sessions *by default*. If you do not want the **Watch Window** contents to persist across Client sessions, clear the **Restore watches on startup** checkbox accessed from the **Preferences** option in the **File** menu.

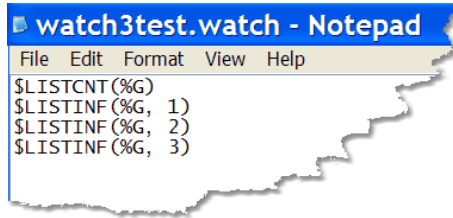
3.2.1.2 Saving and restoring Watch Window contents

If you anticipate watching the same or many of the same items in subsequent debugging sessions, you can save to a local file the entire list of items displayed in the **Watch Window**, then restore those items to the **Watch Window** whenever you want. This may significantly minimize the effort of repopulating the **Watch Window**.

For example, say you are watching the function specifications `$LISTCNT(%G)`, `$LISTINF(%G, 1)`, `$LISTINF(%G, 2)`, and `$LISTINF(%G, 3)`. To save them for later use:

1. Select **Save Watch** from the Debugger Client **File** menu.
2. Specify a workstation folder location and a name for the storage file.

The list of watched items (only) is stored in a text file with a `.watch` file name extension:



Alternatively, you can run the [saveWatch](#)^[246] command from a [mapped](#)^[285] Client button, key, or macro that has the same effect as the above two steps.

Restoring

To restore (at any time) to the **Watch Window** these or other items from any `.watch` file (including any you create independently):

1. Select **Load Watch** from the **File** menu.
2. Locate and select the `.watch` file in the Windows **Select Watch File** dialog box.

[By default](#)^[296], the search for the `.watch` file begins with the folder specified in the `stateFileFolder` element in the `debuggerConfig.xml` file.

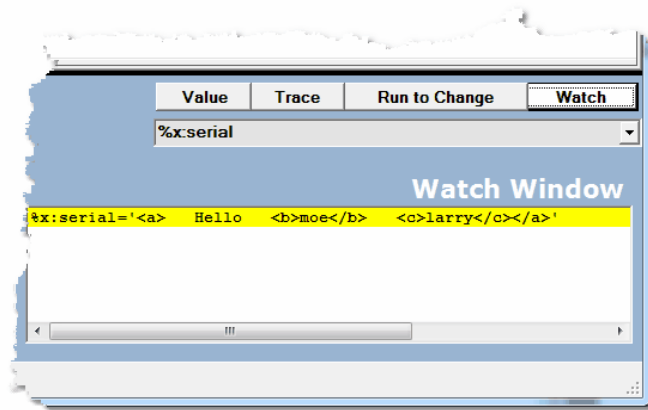
The file's contents are added to the **Watch Window** display with whatever current value they may have in the current source code.

Again, as an alternative, you can run the [loadWatch](#)^[220] command from a [mapped](#)^[285] Client button, key, or macro that has the same effect as the above two steps.

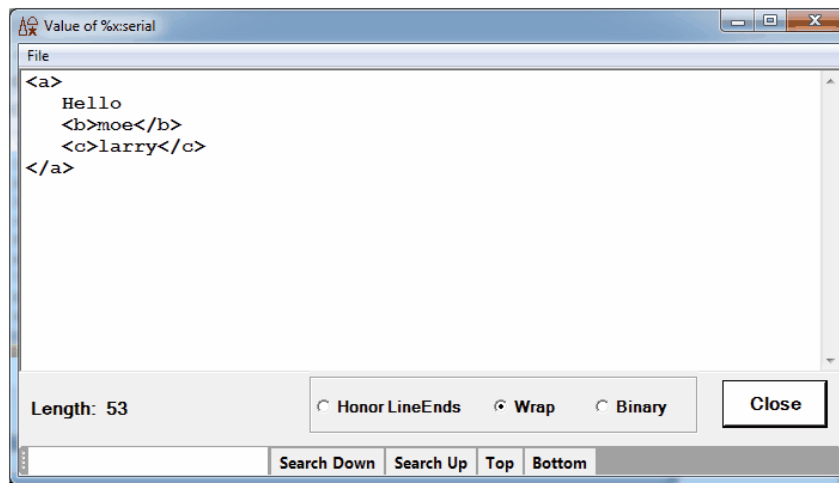
You can save as many watched lists as you like.

3.2.1.3 Getting a detailed view of the value of a watched item

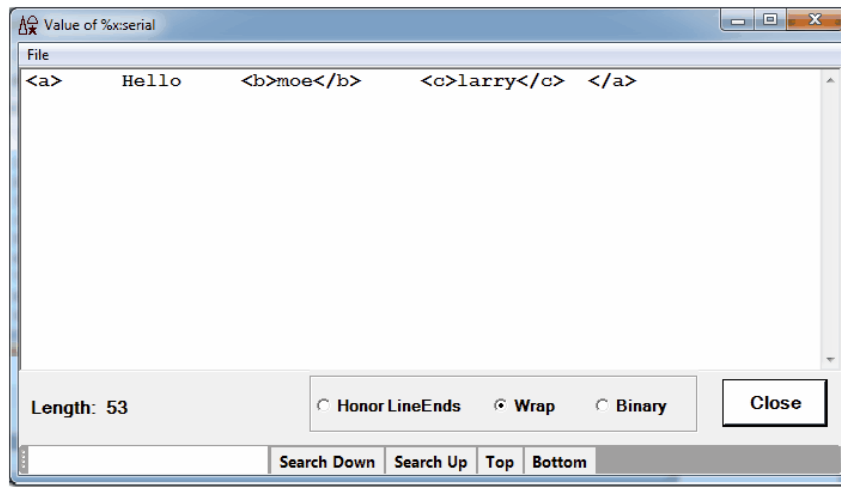
You can double-click an item in the **Watch Window** to display a detailed value window. For example, if you double-clicked **%X:SERIAL** (whose value is a single string that may include line-ends) in the **Watch Window** box shown below:



The value is displayed in a separate window:



By default, any line-end characters in the value are honored in this **Value** window, and the value is broken into lines accordingly. To see the value unbroken at line ends, you click the **Wrap** radio button:



From wrap-view mode, you can always revert to line-end mode by clicking the **Honor Line Ends** button.

To see the hexadecimal values of the data, click the **Binary** button.

To print or save the value, use the **Print** or **Save** options of the **File** menu. The **Save** option saves the file in the window's current display format (ordinary text or hexadecimal digits). To search the value display, use the search bar on the bottom of the window.

Note: As an alternative way to get the same detailed view of a variable that double-clicking it in a **Watch Window** provides, you can specify the variable name in the text box above the **Watch Window**, then click the **Value** button below it. As described in [Displaying temporarily the value of a program data item](#)^[101], a variation of this alternative approach is also the way to get detailed views of the values of \$list, Stringlist object, and XmlDoc object variables — which you **cannot get** by double-clicking them in the **Watch Window**.

See Also

[Adding and removing Watch Window items](#)^[88]

[Displaying temporarily the value of a program data item](#)^[101]

3.2.1.4 Watching Model 204 fields

When the program you are debugging is in a *Model 204* record context, you can view the value of any visible database field in that record by explicitly adding the field name to the **Watch Window**. The field name requires a special prefix, as described below.

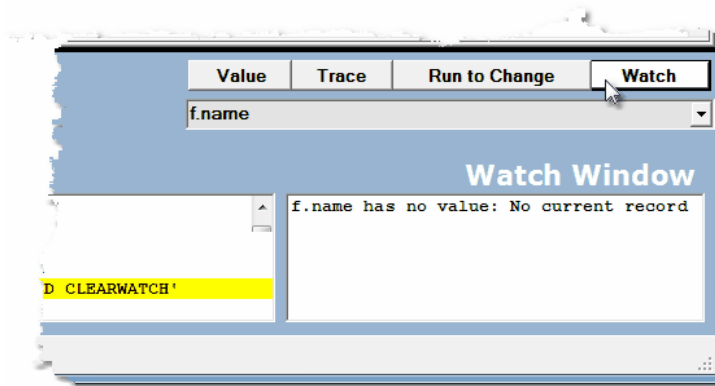
If a field name is referenced within an expression in an HTML or TEXT block in your code, you can also add it to the **Watch Window** (or display it in a **Value** window) by right-clicking the code line that contains it, as described below.

In addition, if a multiply occurring field is in a FOR EACH OCCURRENCE loop, you can [display its occurrence number](#)^[116]. You can also display the values of all the fields in the [current record](#)^[117] or in the [current field group](#)^[120].

Adding fields to the Watch Window

To watch the value of a *Model 204* field:

1. Type its name preceded by **F.** or **f.** (for example, **f.name**) in the [Entity-name input box](#)^[51].
2. Click **Watch**:



By default, the value of the first occurrence of the field is shown in the **Watch Window**; however, you may select later occurrences by subscripting the field name. For example: **f.name(2)** watches and displays the second occurrence of the field name, while **f.name** is equivalent to specifying **f.name(1)**.

Watching fields that belong to field groups

If you are using the *Model 204* field group feature under version 7.6 or higher of the *Sirius Mods*, you can watch or display the value of fields in the current field group by preceding their name by **F.** or **f.**

The execution context must be a current field group (that is, within an FEO FieldGroup loop).

Watching fields specified within HTML or TEXT blocks

You can view the value of a *Model 204* field that is specified as an expression within a User Language HTML or TEXT statement block. In such blocks, the Client detects a field name enclosed by opening and closing braces (for example, `{FIELDNAME}`), and it shows you the field value if you select the **Add Watch** or **Display** right-click option for a line that contains the expression.

See Also

[Displaying the current occurrence value in an FEO loop](#)^[116]

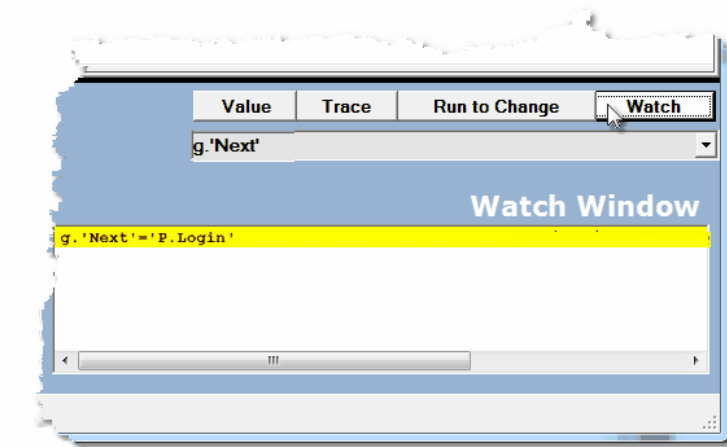
[Displaying all fields in a record](#)^[117]

3.2.1.5 Watching global variables

Like *Model 204* fields, watched global variables must be specified using a prefix. To watch a global variable:

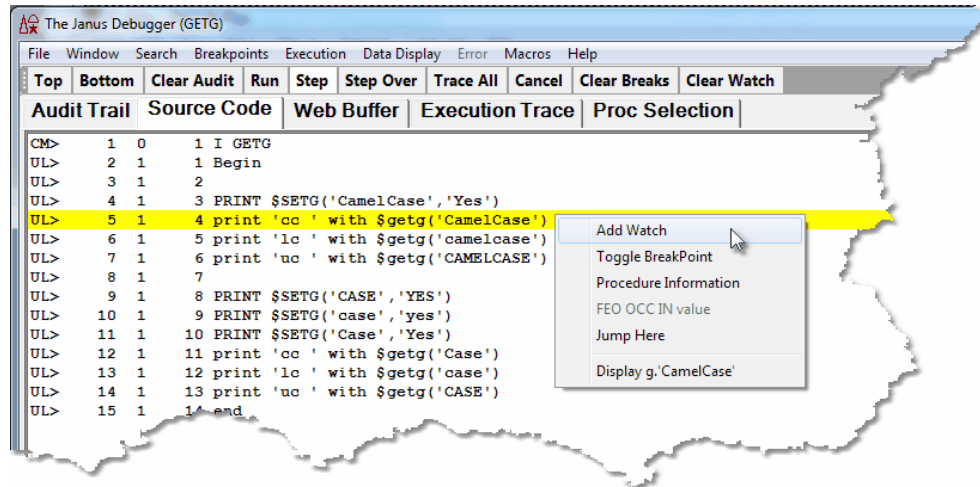
1. In the [Entity-name input box](#)^[51], specify **G.** or **g.** followed by the variable's name within single quotation marks. For example: `g. 'Next'`
2. Click the **Watch** button.

The current value of the variable is displayed in the **Watch Window**:



Note: For global variables whose names are all uppercase, you can omit the single quotation marks enclosing the name. In this case, the Debugger searches for (only) the variable with the all-uppercase form of the name. Whether you specify `g.next` or `g.Next` or `g.NEXT`, the value of only the global variable `NEXT` is displayed.

For a global variables that is referenced by a \$GETG call in a source code statement, you can also right-click its program line and select **Add Watch** from the context menu.



The global variable is added to the **Watch Window** and its current value is displayed.

To view the variable value without adding it to the **Watch Window**, you can right-click the program line and select the **Display** option. The value is shown in a separate **Value** window.

Note: Just as it recognizes \$GETG calls, the Client also recognizes and evaluates \$STATUS, \$STATUSD, and \$CURREC calls as of *Sirius Mods 7.6*.

See Also

[Watching Model 204 fields](#)^[95]

[Viewing dummy string variables](#)^[127]

3.2.1.6 Watching object variables

You add a *Janus SOAP* object variable to the **Watch Window** using either of the ways described [earlier](#)^[88]: right-click the line of code in which it's contained and select the **Add Watch** option, or type its name in the text box above the **Watch Window** and click the **Watch** button.

But for shared objects, class variables within a class definition, and Stringlist and XmlDoc objects, you need to use variations of these techniques. Otherwise, for Stringlist and XmlDoc objects, the Debugger only informs you whether or not the object has content. And for shared objects, the Debugger may fail to "find" the object.

Handling shared objects and class variables are described below. Stringlist objects are discussed in [Watching \\$lists and Stringlists](#)^[98], and XmlDoc objects are discussed in [Displaying Janus SOAP XML document objects](#)^[106].

Watching shared objects

The simplest way to add a shared object variable to the **Watch Window** is to right-click its source code line and add it. When you do so, the class-name qualifier is automatically added as a prefix for the variable. For example, if the line you select on the **Source Code** page is:

```
Print %(tester):sharedPubNum
```

The variable that appears in the **Watch Window** is `%(tester):sharedPubNum`, and you can successfully watch the shared variable's value as you step through the program.

However, if you choose to add the shared variable by first typing its name in the [Entity-name input box](#)^[51], you must be sure to prefix the variable with its class name, explicitly specifying:

```
%(tester):sharedPubNum
```

If you specify only the variable name (`%sharedPubNum`), the variable is added to the **Watch Window** as is, with no `%(tester)` class name prefix. The Debugger does not recognize this as the shared variable in your program, and a "not found" message eventually displays in the **Watch Window**.

See Also

[Displaying temporarily the value of a program data item](#)^[101]

3.2.1.7 Watching \$lists, Stringlists, and Arraylists

To inspect Sirius \$lists, Stringlists, and Arraylists, you apply additional \$functions or object methods to the \$list, Stringlist, or Arraylist variable in the text box above the **Watch Window**. Otherwise, the Debugger will merely report whether or not these variables have content.

This section describes how to watch individual list items or list counts. Using [a related technique](#)^[104], you can display all the list items and their values at once.

To watch Sirius \$lists, Stringlists, or Arraylists:

- For Sirius \$lists, specify either of these \$functions in the [Entity-name input box](#)⁵¹:

<code>\$listcnt(listID)</code>	Gets the number of items in the list
<code>\$listinf(listID, subscript)</code>	Gets the value of a particular list item.

Where:

- `listID` is a Sirius \$list identifier, most often a Float variable. It must be a simple variable or a constant; it may *not* be an image item or a %variable array item, for example.
 - `subscript` is a 1-based list-item subscript that identifies the item to watch.
- For Stringlists or Arraylists, specify either of the following in the Entity-name input box for the %list Stringlist or Arraylist object variable:

<code>%list:count</code>	Gets the number of items in the list
<code>%list:item(num)</code>	Gets the value of a particular list item.

Where:

- `%list` is a Stringlist or Arraylist object variable.
- `num` is a 1-based positive integer that identifies the item to watch. It must be a simple variable or a constant; it may *not* be an image item or a %variable array item, for example.

For example, here are the results of watching the function specifications `$LISTCNT(%G)`, `$LISTINF(%G, 1)`, `$LISTINF(%G, 2)`, and `$LISTINF(%G, 3)`:

Watch Window

```

$LISTCNT(%G)=2
$LISTINF(%G,1)='All work and no play'
$LISTINF(%G,2)=' makes Jack a dull boy.'
$LISTINF(%G,3) has no value: Invalid item number

```

The contents of the **Watch Window**, above, result from evaluating the following test program:

```
INCLUDE LISTTEXT
b
  %i is float
  %g is float
  %g = $listnew
  %i = 777
  $listadd(%g, 'All work and no play')
  $listadd(%g, ' makes Jack a dull boy.')
  $list_print(%g)
end
```

3.2.1.8 Watching class member Variables

In much the same way as for ordinary %variables, you can use the Client's [watch or display](#)^[101] facilities for viewing class member Variables (that is, a variable that is a member of a class, not an instance of a class). If you right-click a code line that is outside the definition of the class, and you use the **Add Watch** or **Display** options, the Client shows the class Variable *and* the object to which it is applied in the **Watch Window** or the **Value** window's title.

If you choose to add the Variable to the **Watch Window** by first specifying it in the text box above the **Watch Window**, you must precede the Variable name by the object variable to which it is applied.

The Client also has [a related feature](#)^[111] that lets you display at once the names and current values of all the Variables in a given class.

Watching Variables within a class definition

To watch the value of a class Variable within a method *within* the class definition, it may be necessary to add it via the [Entity-name input box](#)^[51], rather than by right-clicking its source code line.

The following example of a **Source Code** program helps to demonstrate how to work with **%this** in *Sirius Mods 6.x* versions of the Debuggers. In more recent Debugger versions, class Variables are detected as such, and their value is displayed in the **Watch Window** whether or not you have explicitly preceded the variable name with **%this**.

In the example, the **celsius** variable is referenced in the **fahrenheit** property definition. The class definition is valid and works as intended: both **%celsius** and **%this:celsius** are valid formats for referencing **celsius** in this class definition context.

```

Begin
class thermometer
  public
    variable celsius is float
    property fahrenheit is float
  end public

  property fahrenheit is float
  get
    return (1.8 * %celsius) + 32
  end get
  set
    %this:celsius = (%fahrenheit - 32) / 1.8
  end set
end property fahrenheit
end class

%temp is object thermometer
%temp = new
%input is float
%input='33'
%temp:fahrenheit = %input

print 'Temperature fahrenheit: ' %temp:fahrenheit
print 'Temperature celsius: ' %temp:celsius
End

```

In the Client, you can right-click and select the **Add Watch** menu option to add both `%celsius` and `%this:celsius` to the **Watch Window**. However, as you step through the program, the pre-Mods-7.0 Debuggers will only find and display the value for `%this:celsius`.

In cases like this where class definition code does not explicitly specify `%this`, you must provide the `%this` explicitly yourself by typing it in the Entity-name input box followed by a colon, followed by the class variable name. Then you click the **Watch** button to add the variable to the **Watch Window**.

See Also

[Displaying all Variables of an object's class](#) 

3.2.2 Displaying temporarily the value of a program data item

While the Debugger Client is evaluating program code in the **Source Code** tab, you can view the value of a code variable or field (one time), without adding it to the **Watch Window** for continuous watching. You use either of the following ways:

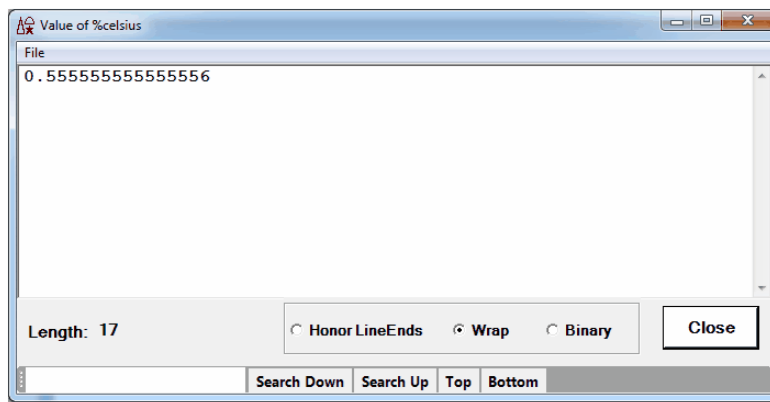
- Right-click the **Source Code** line in which the variable (say, `%var`), is referenced or declared, then select the resulting "Display `%var`" menu option.

Or:

- Type its name in the Entity-name input box below the main window; then click the **Value** button.

Note: For some types of variables, you also need to type the name of special Client functions that produce the appropriate display. Such variables include [\\$list](#)^[104], [Stringlist or Arraylist object](#)^[104], and [XmlDoc object](#)^[106]. And for [class member Variables](#)^[111], you select a right-click option from the **Watch Window**.

Either of the preceding approaches displays the `%var` variable value in a separate **Value of %var** window:



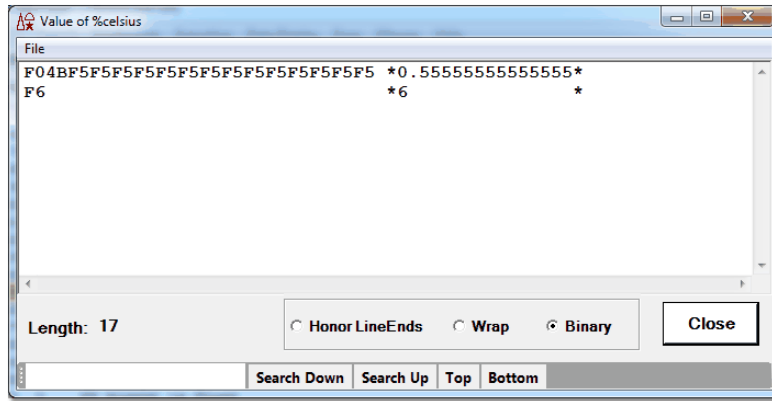
Note: If you have the [console](#)^[316] open, the value is displayed there instead of in a **Value** window. To override this default, use the `valueDisplayOnConsole` option of the Client [setPreference](#)^[260] command.

By default, any line-end characters in the value are honored in the **Value** display window, and the value is broken into lines accordingly.

- To see a value unbroken at line ends, you click the **Wrap View** button. From wrap-view mode, you can always revert to line-end mode by clicking **Honor Line Ends**.
- To see the raw binary representation of the value (16 bytes per line), click the **Binary** button.

This option is useful if the data is not printable (for example, whitespace characters), or if you simply want to see the pre-translation EBCDIC values of the data (since the Debugger Client normally converts strings to ASCII for display).

The binary display shows any printable characters to the right of the hex data, enclosed by asterisks. Non-printable characters are represented by periods:



Clicking either **Honor Line Ends** or **Wrap** restores the original ASCII representation.

- To print or save the value, use the **Print** or **Save** options of the **File** menu. The **Save** option saves the file in the Value window's current display format (ordinary text or hexadecimal digits) in the location you select. To search the value display, use the search bar controls on the bottom of the window.

Note: The [valueDisplay](#)^[279] mappable Client command has the same effect as the **Value** button for the value in the **Watch Window** text box.

See Also

[Getting a detailed view of the value of a watched item](#)^[93]

[Displaying \\$lists, Stringlists, and Arraylists](#)^[104]

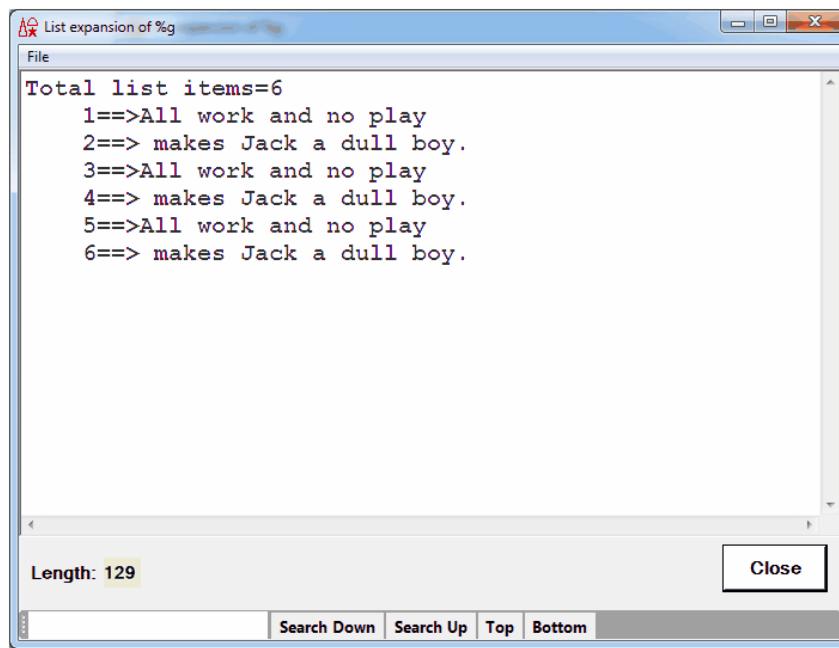
[Displaying Janus SOAP XML document objects](#)^[106]

[Displaying all Variables of an object's class](#)^[111]

[Displaying the current occurrence value in an FEO loop](#)^[116]

3.2.2.1 Displaying \$lists, Stringlists, and Arraylists

You can use the Client's [display](#)^[101] or [watch](#)^[98] facilities for \$lists, Stringlist objects, or Arraylist objects to view or watch an individual item or the count of items. However, when a quick view of the entire \$list, Stringlist, or Arraylist is important, you can access a special type of **Value** window display to see all the list items and their values at once. For example:

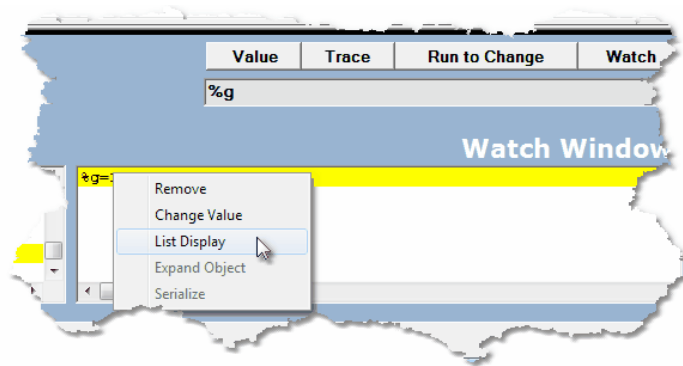


You can also print the window's contents by accessing the **File** menu's print options, and a search bar on the bottom of the window provides controls for searching the window content.

To access the **Value** window from the **Watch Window**:

1. [Add](#)^[88] the variable to the **Watch Window**.
2. In the **Watch Window**, right-click the variable.

If this is a list variable and its value is currently non-Null, the **List Display** context menu option is enabled:

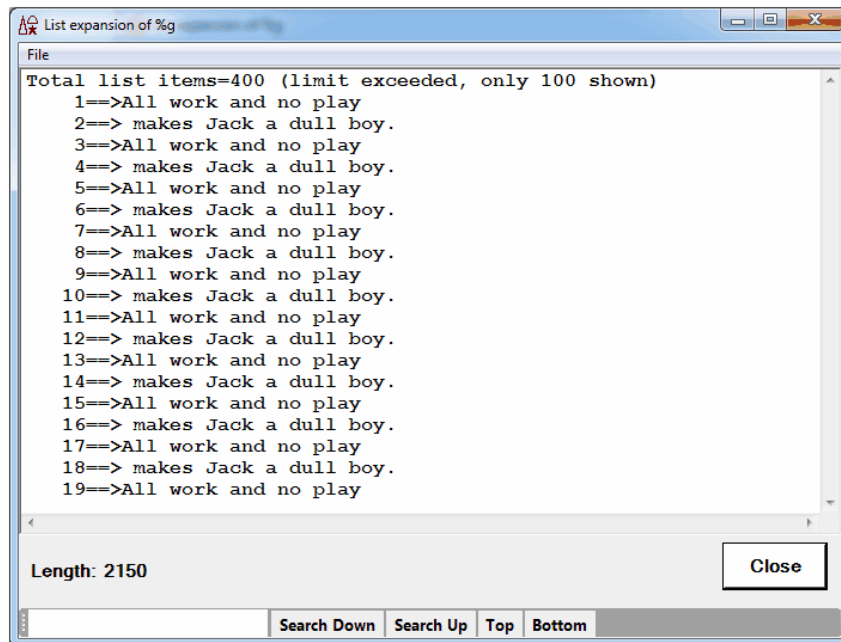


3. Select **List Display**.

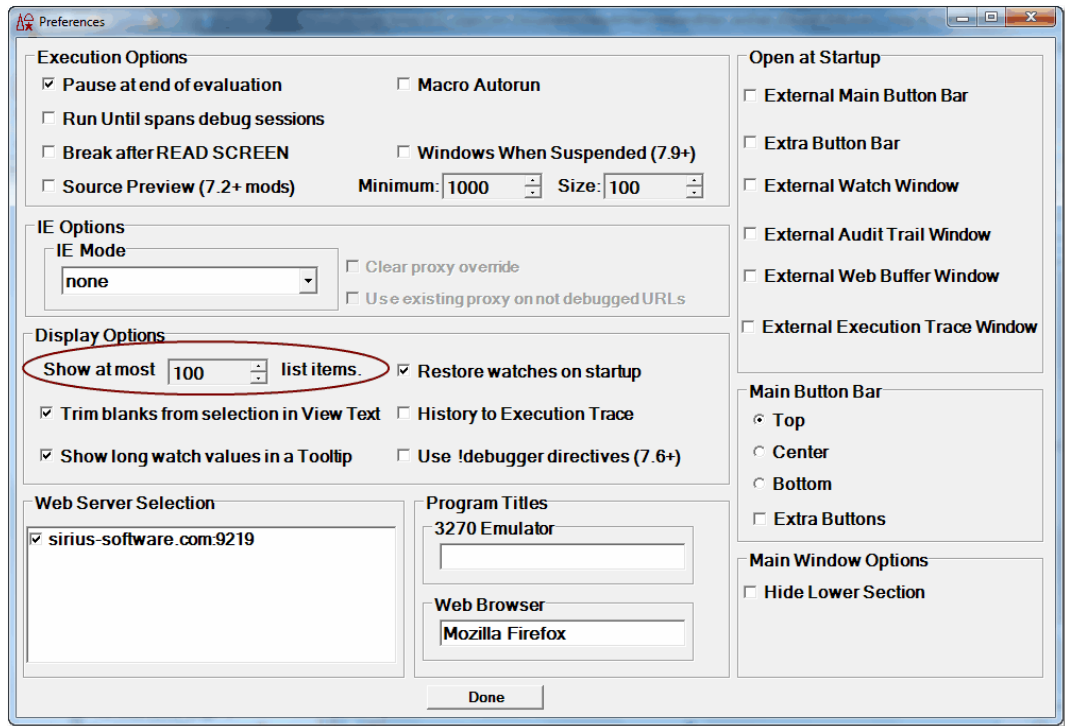
Note: For this feature, subscripted references are allowed for \$list variables. For example, to display \$list item *i* in array %L, add %L(%i) to the **Watch Window**.

Note: The `expandList` [mappable Client command](#)^[285] has the same effect for its argument as clicking **List Display** for a watched item.

By default, a limit of 100 items is enforced. If there are more items than the limit, this is noted in the “Total list items” line, and only the first 100 are displayed, as shown below:



If the limit of 100 is too low, you can reset it in the **Display Options** area in the **Preferences** dialog box (accessed via the **Preferences** option of the **File** menu):



The limit may be set in the range from 100 to 10,000, in increments of 1000.

See Also

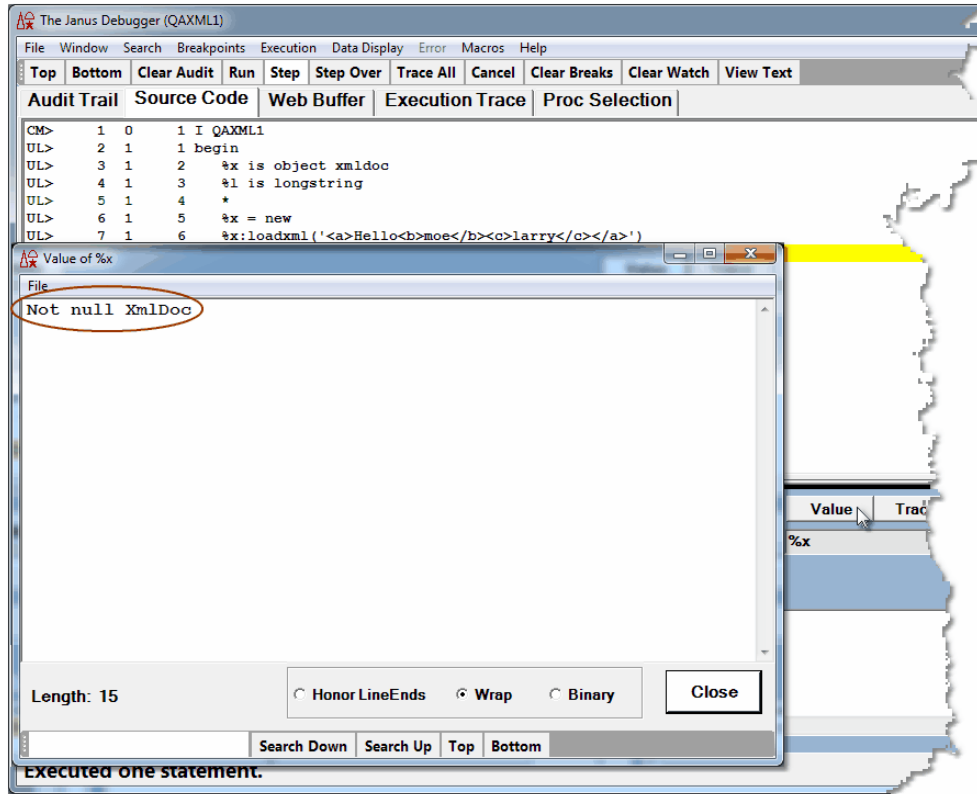
[Watching \\$lists, Stringlists and Arraylists](#) ⁹⁸

3.2.2.2 Displaying Janus SOAP XML document objects

Applications using *Janus SOAP XML* document objects are an increasingly common type of *Janus Web* application. Ordinarily, however, to inspect the contents of an *XmlDoc* or *XmlNode* object, it must be serialized by one of the XML document serializing methods like *Serial*, *WebSend*, *Xml*, or *Print*. And in either the *Janus Debugger* or the *Sirius Debugger*, if you are debugging a request in which a line like the following is executed:

```
%x:loadxml(' <a>Hello<b>moe</b><c>larry</c></a>')
```

Requesting¹⁰⁷ the value of `%x` merely reports *whether* the object has content:



If the program in the preceding example contained a subsequent assignment statement to a string variable like the following, you could display the XmlDocument content by watching for or requesting the value of `%longstring`:

```
%longstring = %x:serial(, 'EBCDIC')
```

For parts of the code where no such serializing statement is present, however, the Debugger **Serialize** feature lets you implicitly call the *Janus SOAP* XML document Serial method to view the contents of an XmlDocument or XmlNode object variable.

You can invoke the feature multiple ways for XmlDocument variable `%doc`:

- Display the expanded value of `%doc:serial` in a **Value** window.
 - a. Enter `%doc:serial` in the Entity-name input box below the main window.
 - b. Click the **Value** button.
- Or:
 - a. Add `%doc` to the **Watch Window** (by right-clicking the **Source Code** line containing `%doc` and selecting the **Add Watch** option, or by entering `%doc` in the text box above the **Watch Window** and clicking the **Add Watch** button).

- b. Right-click `%doc` in the **Watch Window** and select the **Serialize** option.
- Display the non-expanded value of `%doc:serial` in the **Watch Window**:
 - a. Enter `%doc:serial` in the Entity-name input box.
 - b. Click the **Watch** button.

For the example code above, if you specify `%x:serial` in the Entity-name input box and click the **Value** button (at any point after the execution of the `%x:loadxml` statement cited at the beginning of this section), the following **Value** window displays:



The Debugger **Serialize** feature formats the return value of the **Serial** call as if you specified the following form of the method (which calls for a single string with EBCDIC characters, added carriage-return/line-feed character sequences, and added three-blank character sequences for the indent of nested elements):

`Serial(, 'EBCDIC CRLF Indent 3')`

As described in the *Janus SOAP Reference Manual*, the *Janus SOAP* **Serial** method defaults are **UTF8** instead of **EBCDIC**, and no added line-end or indentation characters. So for user convenience, the Debugger is internally providing its own **Serial** method defaults (which you can adjust, as described in the ["Serial method formatting defaults"](#)^[106] subsection).

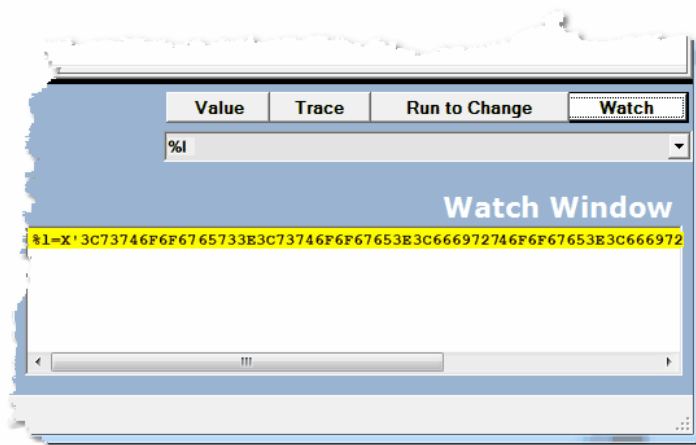
Debugger Serial method compared to source code Serial methods

The Debugger's internal Serial method handling is independent of Serial method calls actually specified in the source code you are debugging. For example, in the following excerpt, the User Language object pointed to by `%d` (of type `document`) contains a variable member named `x` that is an `XmlDoc`:

```
class document
  public
    variable x is object XmlDoc
    constructor new
      subroutine addStooge(%iFirst is longstring, -
                          %iLast is longstring)
    end public
  ...
  %d is object document
  %d = new
  %l is longstring

  %d:addstooge('Moe", 'Howard')
  ...
  %l = %d:x:serial
  ...
```

If you are watching `%l`, which is not an object variable, you find that its value is shown as an Unicode binary string when the `%l = %d:x:serial` statement in the request is executed:

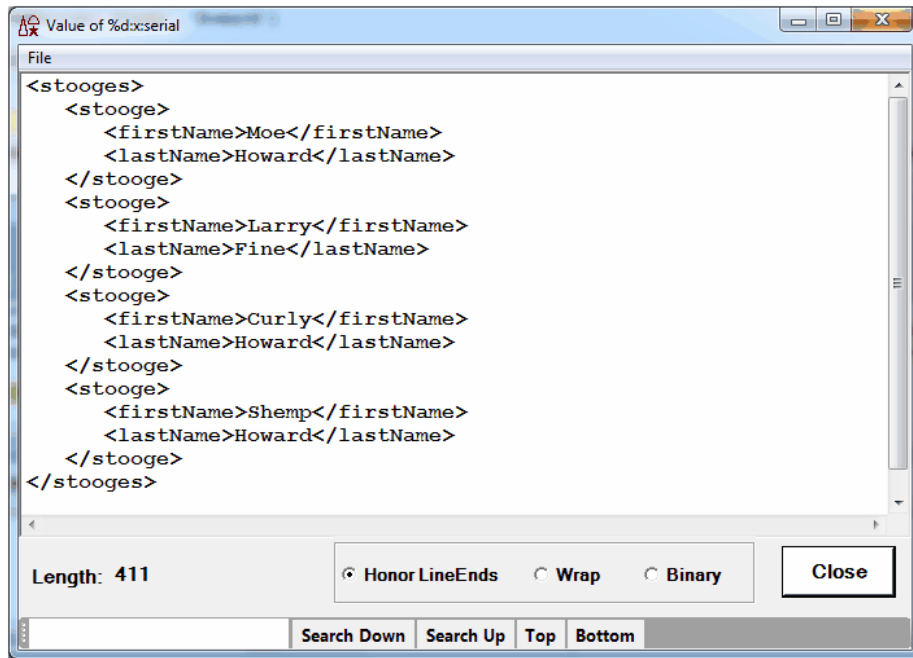


However, you can use the Debugger's Serial method feature and request a detailed view of the `XmlDoc` object variable `x`:

1. Specify `%d:x:serial` in the Entity-name input box below the main window.

2. Click the **Value** button.

You see the serialized form of the XmlDocument in the **Value** window:



Serial method formatting defaults

The preceding **Value** window display also shows again the effect of the implicit Debugger Serial parameters CRLF and Indent 3. These formatting defaults (for XmlDocument and XmlNode objects) are specified in the Debugger [Client configuration file](#)^[374] (debuggerConfig.xml):

```
<serialParms>CRLF INDENT 3</serialParms>
```

You can change this formatting, for example, to increase the indentation:

```
<serialParms>CRLF INDENT 10</serialParms>
```

To suppress this feature (never have the Debugger add parameters to serial), specify no value:

```
<serialParms></serialParms>
```

See Also

[Watching object variables](#)^[97]

3.2.2.3 Displaying all Variables of an object's class

Although you can view the contents of *Janus SOAP* Stringlists and XmlDocs, the Debugger cannot display the entire contents of most *Janus SOAP* objects. However, you can display for a given object variable the names and values of the Variables (public and private) or Variable-like members defined for its class.

For user classes, members defined as Variables are displayed. For system classes, members that are similar to Variables, that is, that take no arguments (certain Functions and Properties) and return simple values (names, counts, positions, etc.), are displayed.

This feature is supported for *Janus SOAP* system classes (including system exception classes) and for user-defined classes, *excluding* these *Janus SOAP* classes:

System and Subsystem
FastUnloadTask
DebuggerTools

For any class, the display does *not* include class methods (Functions, Subroutines, Properties, or Constructors).

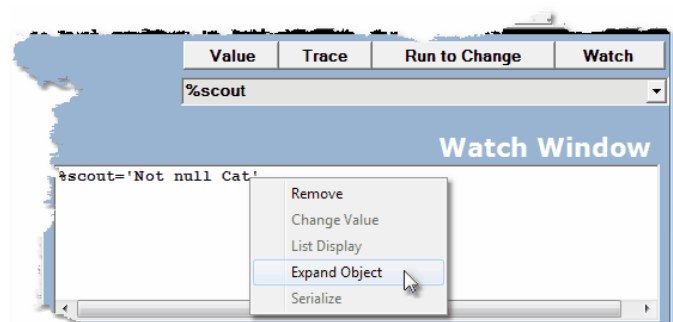
The Variable-like members you can display for the Sirius system classes are [summarized](#)^[113] later in this section.

The `expandObject` [mappable Client command](#)^[285] also invokes this feature, and the `getVariablesForClass` command invokes a variant of the feature that retrieves variable names but not values.

To display the member variables in the class represented by a particular object variable:

1. [Add](#)^[88] the object variable to the **Watch Window**.
2. In the **Watch Window**, right-click the object variable.

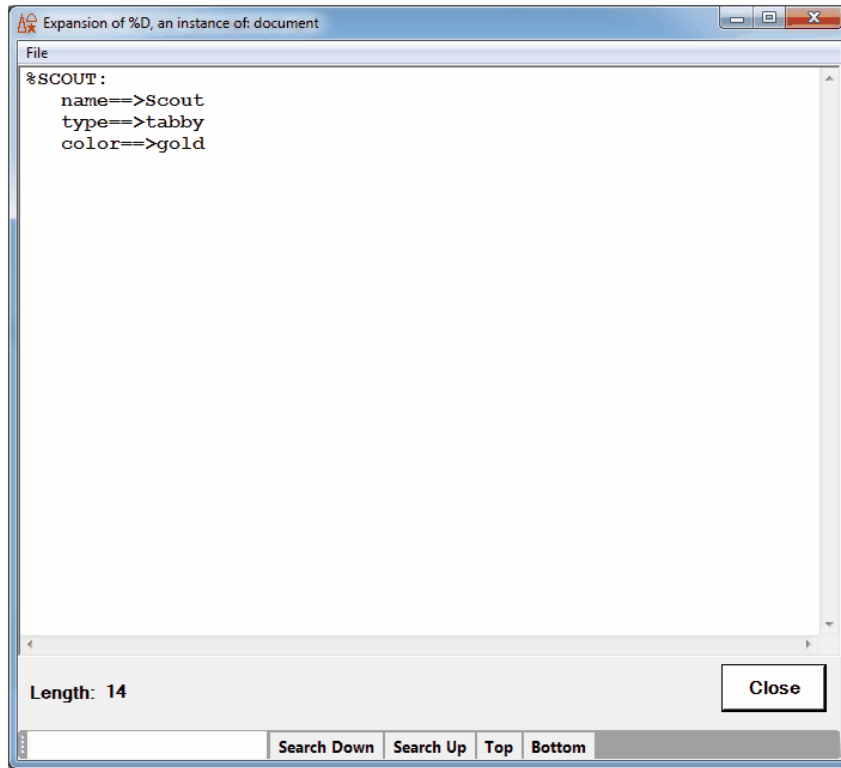
If this is indeed an object variable and its value is currently non-null, the **Expand Object** context menu option is enabled:



If the object variable's value is **Not-null**, its class name is automatically displayed as well. If the current object variable value is other than **Not-null**, no class name appears and the **Expand Object** option is not enabled.

3. Select **Expand Object** if the object variable's value is **Not-null**.

A **Value** window opens, displaying the names and current values of the member variables in the class. For example, a window like the following displays:



This is the class definition for the variable display above:

```
class cat
  public
    variable name is longstring
    variable type is longstring
    variable color is longstring
    constructor new (%iName is longstring)
    subroutine talk
  end public

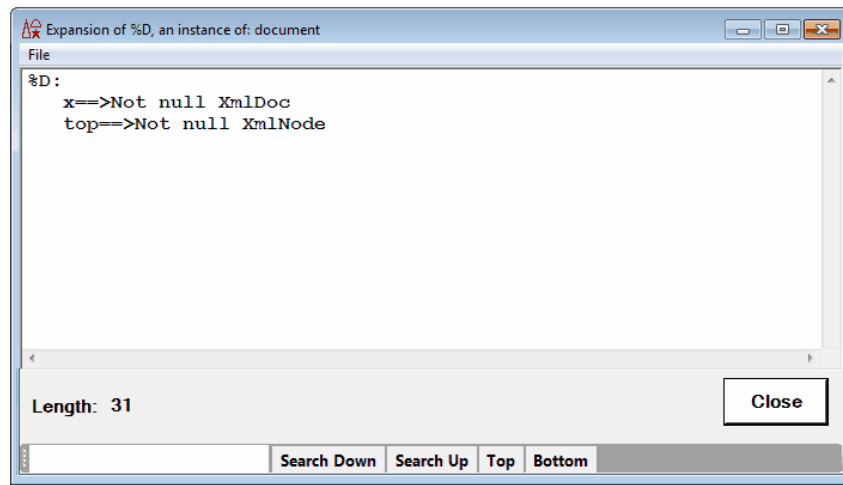
  constructor new (%iName is longstring)
    %this:name = %iName
  end constructor new

  subroutine talk
    print %this:name ' says Meow'
  end subroutine talk
end class cat
```


At the same time the Variables are displayed, a [status bar](#)^[50] message reports the number of Variables in the class. For this example: **Found 3 variables in CAT.**

Note: If the Variables of a class are themselves objects, the only displayable information about their values may be that they are "Not null."

For example, for %d, an instance of a user-defined Document class that contains a *Janus SOAP* XmlNode variable and an XmlNodeList variable, the class Variable display is similar to this:



Summary of variable members in Sirius system classes

If the object variable you expand (as described above) is an instance of a Sirius system class, the members that display are summarized in the table below.

Class	Viewable members
Arraylist	Count LastItem
CharacterMap	<i>none suitable</i>
CharacterTranslationException	BytePosition CharacterPosition Description HexValue Reason
Dataset	State
Daemon	AmDaemon HaveDaemon MasterNumber ParentNumber

Class	Viewable members
DaemonLost	<i>none suitable</i>
FastUnloadTask	State (<i>Sirius Mods 7.6+</i>)
FloatNamedArraylist	Count Default UseDefault
Email	GetReplyCode GetReplyText Host Port
HttpRequest	Fieldcount Headercount Host Httpversion Maxredirects Page Port Protocol Proxy Timeout Url
HttpResponse	Code Content HeaderCount Message StatusLine Success Url
InvalidBase64Data	Position
InvalidHexData	Position
InvalidRegex	Code Description Position
Ldap	ErrorNumber ErrorText
MaxDaemExceeded	<i>none suitable</i>
NamedArraylist	Count Default UseDefault
NoFreeDaemons	<i>none suitable</i>

Class	Viewable members
RandomNumberGenerator	<i>none suitable</i>
Record	FileName (<i>Sirius Mods 7.6+</i>) LockStrength RecordNumber (<i>Sirius Mods 7.6+</i>)
Recordset	IsEmpty LockStrength
RecordsetCursor	FileName (<i>Sirius Mods 7.6+</i>) LockStrength LoopLockStrength (<i>Sirius Mods 7.6+</i>) RecordNumber (<i>Sirius Mods 7.6+</i>) State (<i>Sirius Mods 7.6+</i>)
Screen	ActionKey Columns Rows
ScreenField	Column Invisible ItemId Modified Numeric Protected Row Value Width
Socket	Errinfo('CODE') Errinfo('SOCKNUM') Errinfo('FUN') Info('REMOTE') Info('STAT')
SortedRecordset	IsEmpty
StringTokenizer	AtEnd (<i>Sirius Mods 7.6+</i>) CurrentQuoted (<i>Sirius Mods 8.0+</i>) CurrentToken (<i>Sirius Mods 8.0+</i>) NotAtEnd (<i>Sirius Mods 7.6+</i>) String (<i>Sirius Mods 8.0+</i>) StringLength (<i>Sirius Mods 8.0+</i>)
Stringlist	Count LastItem MaxItemLength
UnicodeNamedArraylist	Count Default UseDefault

Class	Viewable members
UnknownStatistic	Name
UserStatistics	LoginToString (<i>Sirius Mods 7.6+</i>) RequestToString (<i>Sirius Mods 7.6+</i>) ToString (<i>Sirius Mods 7.6+</i>)
XmlDoc	DefaultURI Length LocalName Prefix Qname Type URI Value
XmlNode	DefaultURI Encoding InvalidChar Length LocalName Prefix Qname Type URI Value XPathOrder
XmlNodeList	Count

3.2.2.4 **Displaying the current occurrence value in an FEO loop**

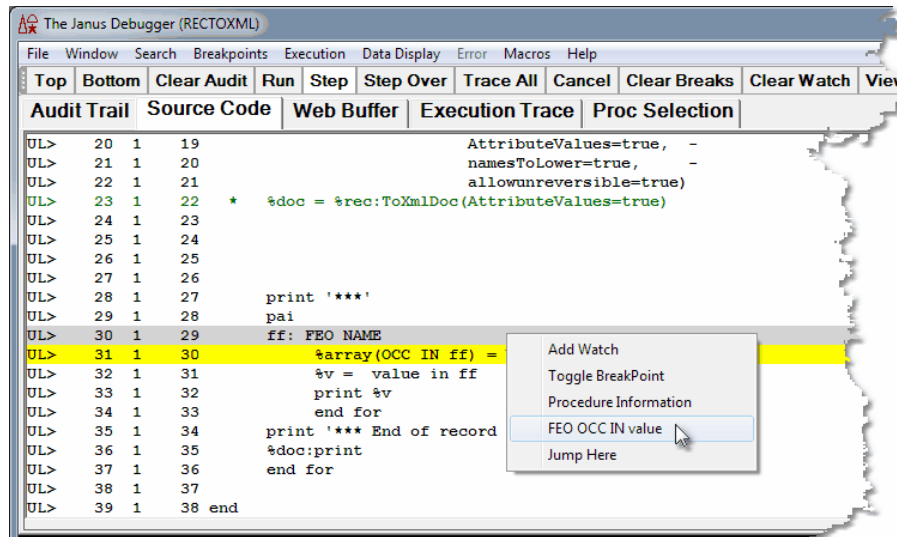
In User Language, the OCC IN phrase lets you refer back to the label of a FOR EACH OCCURRENCE OF (FEO) statement to get the number of the current field occurrence in the loop. For example:

```
ff: FEO MISCINFO
  %array(OCC IN ff) = VALUE IN ff
  ...
```

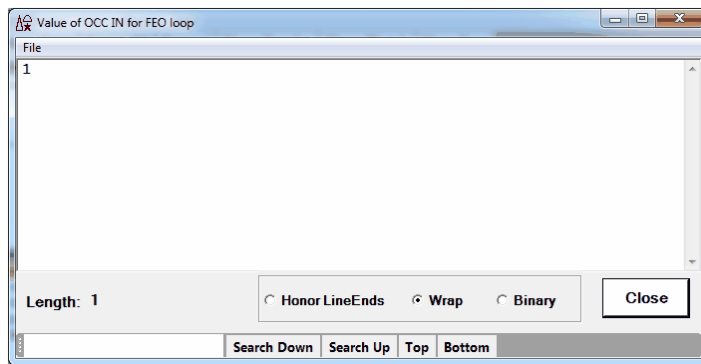
If you are debugging a request that contains FEO loops, you can easily display the current occurrence number value in the Debugger Client:

1. In the **Source Code** page, right-click on a line that contains an FEO statement.

2. Select **FEO OCC IN value** from the context menu:



The occurrence value displays in the **Value of OCC IN for FEO loop** window:



The Client [feoDisplay](#)^[208] command performs the same operation.

3.2.2.5 Displaying all fields in a record

The User Language PAI (Print All Information) statement displays the values of all the visible fields in a given *Model 204* record. You can get this same display from within the Debugger by using the `pai` command.

The `pai`^[235] command is a Debugger Client [command](#)^[285] you specify in a [macro](#)^[309] or you map to a Client button or hot key. If you then issue the command while debugging at an execution point where there is a current record (for example, inside of a record oriented FOR loop), the Client generates a display of the record's fields that is the same as that of the User Language PAI statement.

A typical scenario for using the `pai` command might include these steps:

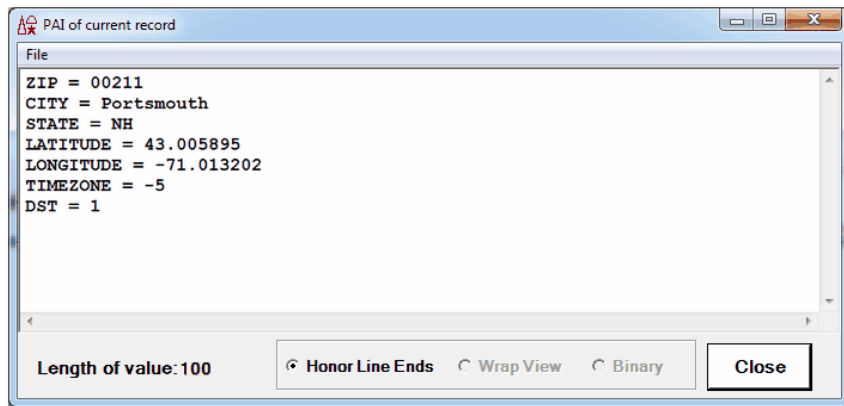
1. You map the command to a Client hot key.

In a [ui.xml file](#)^[287], you specify the following to set up ctrl-a as a hot key to issue a `pai` command:

```
<mapping command="pai" key="a" keyModifier="ctrl"/>
```

2. While debugging a request, you press ctrl-a when you are in a record context.

In a **Value** window, you see a PAI-style display of the current record:

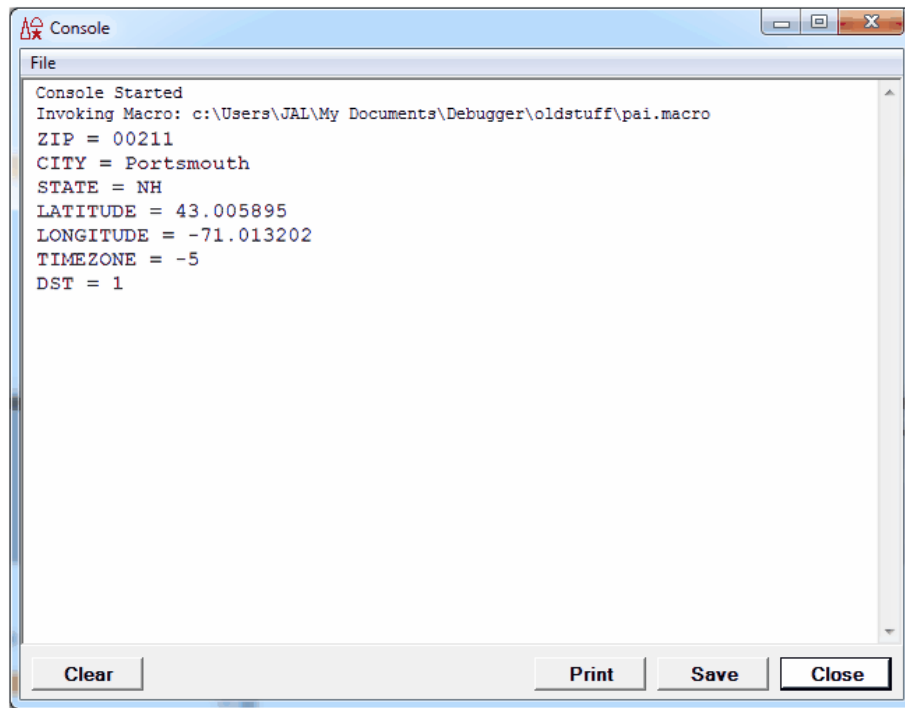


A Value window display is the default. As shown below, the output may also be displayed in the macro console window.

Alternatively, you can use the `pai`^[235] command in a Debugger macro or from the [command line](#)^[316]. For example, say your macro is a `pai.macro` text file that has the single statement:

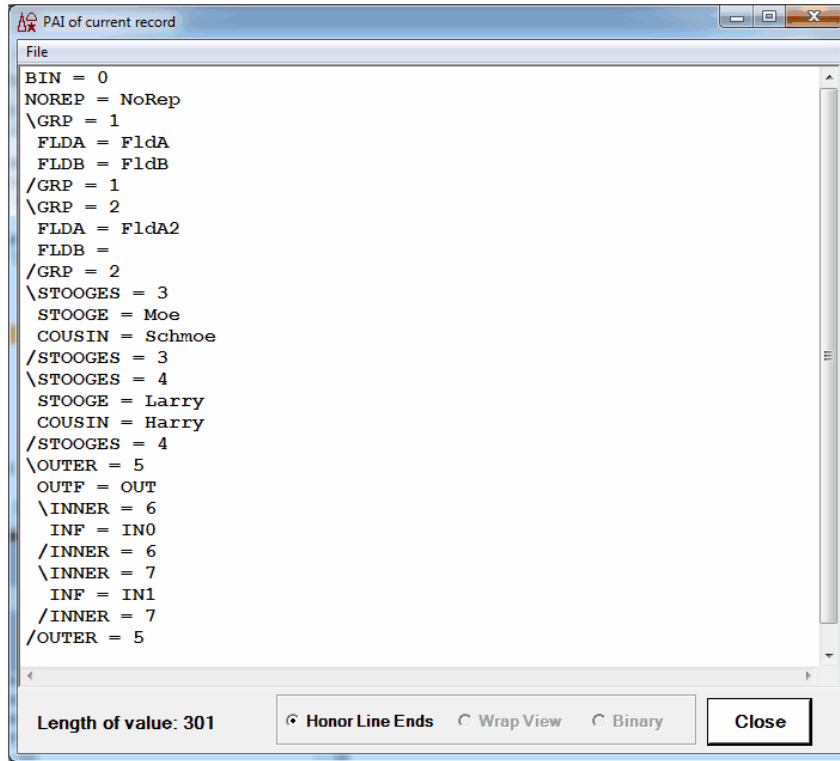
```
pai
```

You [run the macro](#)^[309] at a debugging execution point where there is a current record, and although it is not required, you have the [macro console](#)^[316] open. The record's PAI output displays on the console:



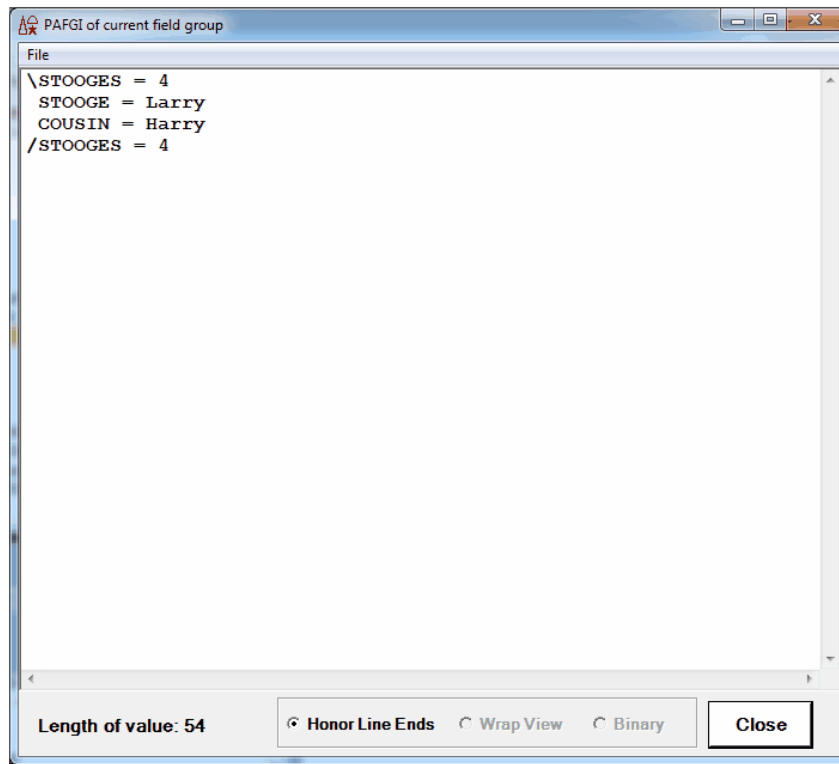
Displaying a record's field groups

If you are using *Model 204 V7R2* or better, and you are debugging a request against a file in which field groups are defined, the `pai` command display of a record also includes the field group occurrences, each beginning with a backward slash (\) and ending with a forward slash (/):



To display only the fields in the current or specified field group, you use the `pafgi`²³⁴ command, which produces the same output as the User Language PAFGI (Print All Fieldgroup Information) statement. The discussion above about ways to map and use the `pai` command applies entirely to the `pafgi` command as well.

Invoking the `pafgi` command while within field group context produces a display of the fields in the current or specified field group. For example, for the `STOOGES` field group in the same request and record as in the preceding example, this is the `pafgi` display:

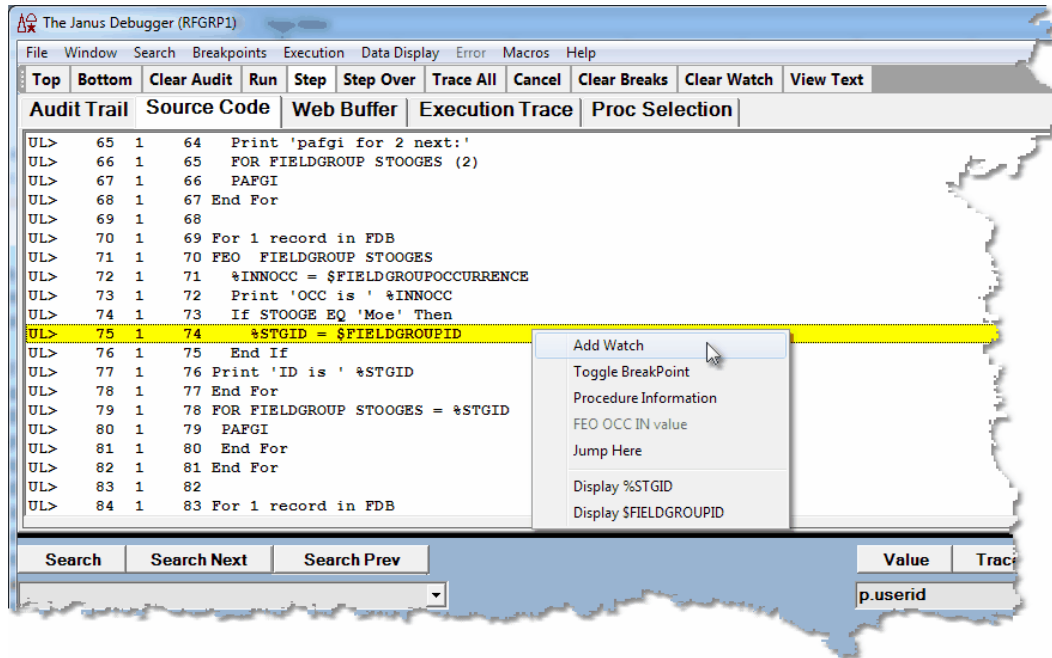


Field group \$functions

The Debugger recognizes and evaluates User Language `$FieldGroupID` and `$FieldGroupOccurrence` function calls. These functions are useful for operating selectively on field group instances.

`$FieldGroupID` returns the unique numeric identifier that *Model 204* assigns to each field group in a record. `$FieldGroupOccurrence` returns the current occurrence number of an occurrence of a repeating field group.

To view the value of either \$function, you can right-click its program line in the **Source Code** page and select **Add Watch** from the context menu.



The \$function is added to the **Watch Window** and its current value is displayed whenever the current program context is a field group. The function value subsequently displays when the program execution point is in any field group.

To view the \$function value without adding it to the **Watch Window**, you can right-click a program line that explicitly contains it and select the **Display** option. The value is shown in a separate **Value** window.

3.2.2.6 Displaying Model 204 parameters

You can use the Client's display or watch facilities to view the value of any legal *Model 204* parameter:

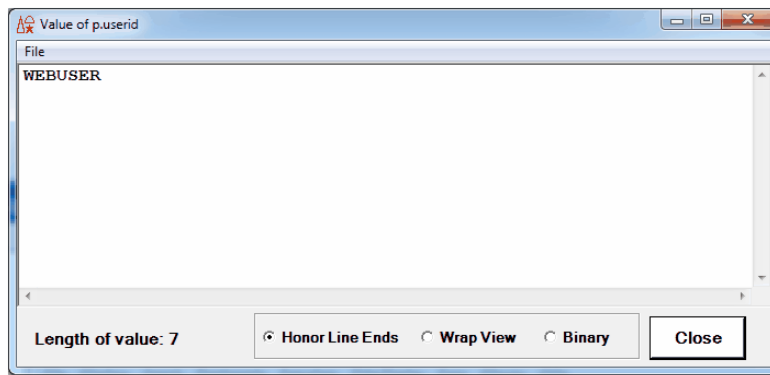
1. In the Entity-name input box below the main window, specify `p.` followed by the parameter's name. For example:

`p.userid`

Case does not matter, nor do leading or trailing blanks.

2. Click the **Value** button above the **Watch Window**.

The parameter value displays in a separate **Value of p.parm** window:



Note: If the parameter is a file parameter, the value of that parameter is returned for the current file. If there is no current file, an error is returned.

If the parameter is a User parameter, the value returned is that for the current web thread (if *Janus Debugger*) or for the current Online user (if *Sirius Debugger*).

An alternative path to the same Value window is via the **Watch Window**:

1. In the Entity-name input box, specify **p.** followed by the parameter's name.
2. Click the **Watch** button above the text box to add the parameter to the **Watch Window**.
3. In the **Watch Window**, double-click the parameter name.

3.2.2.7 Displaying the Universal Buffer content

You can use the Client's display or watch facilities to view the current content of the user's Universal Buffer. The *Model 204* Universal Buffer is used to transport Large Object data and with the MQ/204 interface.

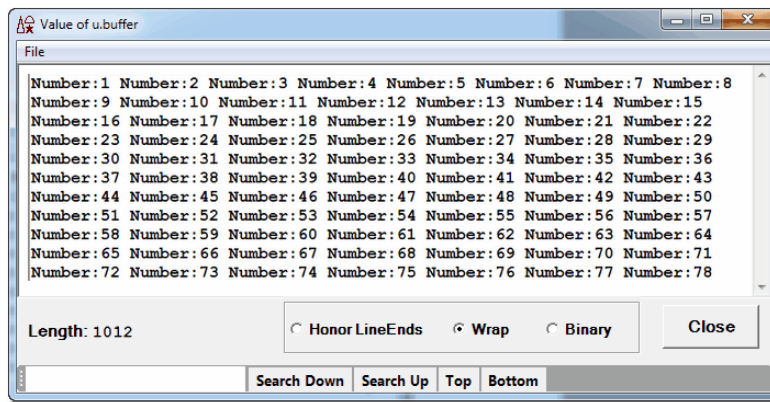
1. In the [Entity-name input box](#)^[51] below the main window, specify the following:

u.buffer

Case does not matter, nor do leading or trailing blanks.

2. Click the **Value** button above the text box.

The buffer content displays in a separate **Value of u.buffer** window:



Features of the window include:

- A **Wrap** button to see a value unbroken at line ends (from which, you can revert to line-end mode by clicking **Honor Line Ends**)
- A **Binary** button to see the raw EBCDIC/binary representation of the value (16 bytes per line)
- Familiar Windows **Print**, **Page Setup**, and **Print Preview** dialog boxes accessible from the viewer's **File** menu
- Buttons to search within the displayed value, as well as buttons to move to the top or bottom of the value.
- Standard mouse-based copying of text data (but altering or deleting text is **not** allowed)

An alternative path to the same Value window is:

1. In the Entity-name input box, specify: **u.buffer**
2. Click the **Watch** button above the text box to add the buffer to the **Watch Window**.
3. In the **Watch Window**, double-click the **u.buffer** string.

3.2.3 Setting the value of a variable

You may set or update the value of certain types of variables with the Janus and Sirius Debuggers. The following types of variable values may be modified:

- String %variables (255 character limit)
- Unicode %variables (UTF-16 bytestreams)
- Float %variables

- Fixed %variables
- Longstrings (255 character limit)
- Global variables
- Screen and Image items
- Janus SOAP Boolean enumeration variables (as of Version 7.7 of the *Sirius Mods*; only True and False [case not important] are valid values)

Currently, only scalar variables may be set; array elements may not be set. In addition, most object variables may *not* be set.

To set the value of a variable with the Debugger:

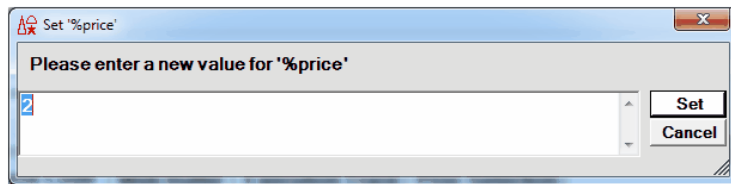
1. [Add](#)^[88] the variable to the **Watch Window**.

2. In the **Watch Window**, right-click the variable.

If this type of variable can be set, and it is settable at this point in the execution of the program, the **Change Value** context menu option is enabled.

3. Select **Change Value**.

A **Set %var** dialog box like the following displays, where %var is replaced by the variable you are modifying, and the variable's current value is displayed:



4. Specify the new variable value, and click **Set** or press the Enter key.

The **Watch Window** is refreshed with the new value, **Variable Set** displays in the [Status bar](#)^[50], and the new value becomes the current value of the variable *in the program* at the current point of program execution.

The [setM204Data](#)^[260] command is equivalent to right-clicking a **Watch Window** item and selecting **Change Value**.

Note: Setting a new value in the Debugger follows the rules of User Language assignment. For example, specifying a text value for a float variable results in setting the variable to zero.

See Also

[Watching program data items](#)^[87]

[Displaying temporarily the value of a program data item](#)^[101]

3.3 Getting source file, audit trail, and web buffer information

These sections are included:

[Locating and editing procedure source files](#)^[126]

[Viewing dummy string variables](#)^[127]

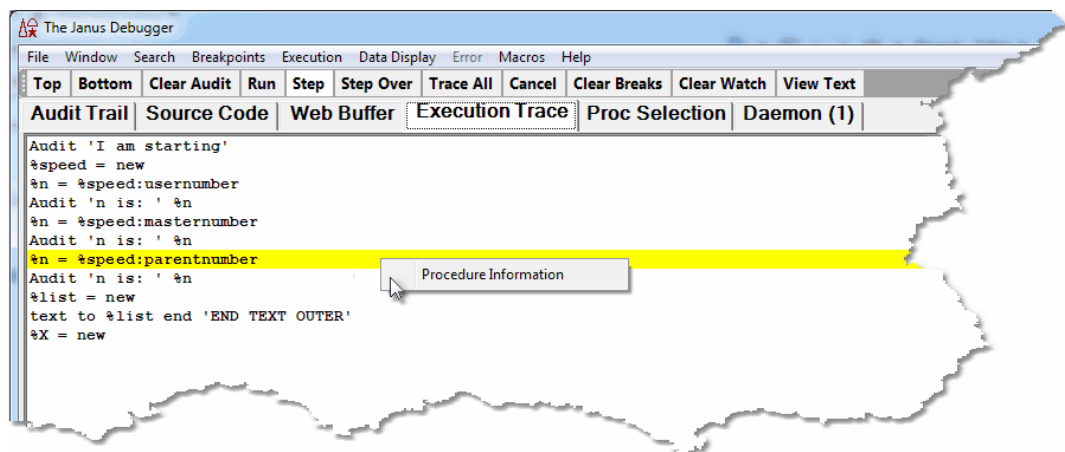
[Viewing the audit trail](#)^[128]

[Viewing the web output buffer](#)^[129]

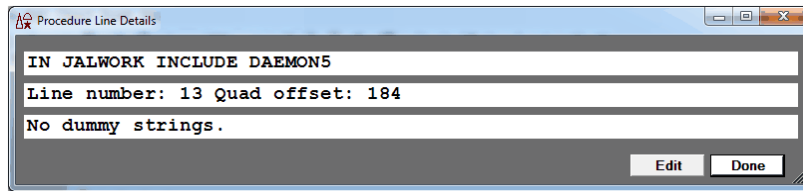
3.3.1 Locating and editing procedure source files

If your application has a complex structure of nested procedure includes, the *Model 204* procedure that contains a particular source code line may not be apparent. The Debugger Client provides a simple way to determine this information:

1. Right-click a code line in the **Source Code**, **Execution Trace**, or a **Daemon** tab to display the context menu's **Procedure Information** option:



2. Select **Procedure Information** to display a **Procedure Line Details** dialog box like the following:



The top line of the dialog box, which identifies the procedure file and procedure name, also identifies the APSY subsystem within which the procedure was invoked, if any.

The middle line in the dialog box identifies the statement's line number within the procedure, and the bottom line in the dialog box displays the original dummy string variables if the statement is the result of a dummy string substitution.

If you click the **Edit** button on the **Procedure Line Details** dialog box, you invoke a local text editor (if you have [configured](#)^[166] a Sirius-supported editor). You then can edit the procedure, and save it back to the Online.

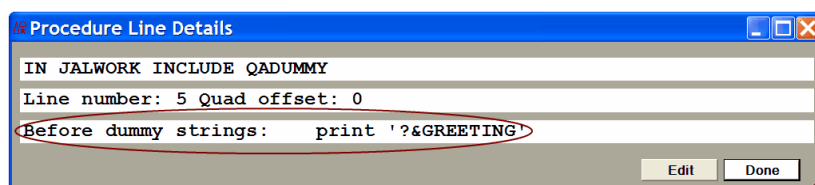
3.3.2 Viewing dummy string variables

If you [invoke](#)^[126] a source code line's procedure details, and the line had ?& (global variable) dummy string substitutions, the text of the original line, *prior* to the substitution of the global variable value for the dummy string, is displayed.

For example, a statement you suspect to have contained a dummy string variable is displayed in the **Source Code** page as:

```
print 'Hello'
```

When you select **Procedure Information** after right-clicking the code line, you see a **Procedure Line Details** display like the following, which identifies the source procedure and shows the dummy string variable name:



If subsequently you want to [watch](#)^[87] the values of this dummy string variable, you must [specify it explicitly with a "g." prefix](#)^[96] (omitting the ?& characters) and add it to the **Watch Window**.

Note: To view dummy string substitutions in Model 204 commands, you must be running at least version 7.6 of the *Sirius Mods*.

See Also

[Locating and editing procedure source files](#)^[126]

[Watching global variables](#)^[96]

[Source Code tab](#)^[11]

3.3.3 Viewing the audit trail

Model 204 audit trail information is available in two places in the Debugger Client:

- The **Audit Trail** tabbed page
- The **Most Recent Audit Trail** window

The Audit Trail tab

This tab displays the *Model 204* audit trail lines produced by the thread that is servicing your web request or executing your 3270/Batch2 request, and by any threads that run daemons on behalf of either of these types of thread. In addition, this page displays information about the state of the Debugger Client, such as the port on which it is listening, and for the *Janus Debugger*, about incoming connections to the web server from other web browsers.

To see audit trail lines that are no longer displayed on the page:

- Click the **Top** button (below the page) to display the beginning of the audit trail, or click the **Bottom** button to display the end of the audit trail.
- Use the scroll bar (to the right of the page), drag a highlighted line toward the top or bottom of the page, or use the **Search**, **Search Next**, and **Search Prev** buttons.

By [default](#)^[291], the F9 key is equivalent to the **Search Next** button. **Search Prev** (or pressing the Alt key while clicking either the **Search** or **Search Next** button) makes the search operate backwards, that is, from bottom to top (the Ctrl+U key combination has the same result).

Pressing the Enter key after clicking **Search Next** (or whenever the **Search Next** button is highlighted) repeats the **Search Next** action. To get this same result you can also press the Ctrl+F key combination (to give focus to the Search text area), then press the Enter key.

To clear the **Audit Trail** display, click the **Clear Audit** button, below the page. The lines from the entire session are removed. Subsequent actions update the display — recording from the moment of the action and not from the beginning of the session, however.

The Most Recent Audit Trail window

This window displays only the last few lines of the audit trail for this web user or 3270/ Batch2 thread. It displays *no* non-audit trail information.

The **Most Recent Audit Trail** display is *not* deleted if you click the **Clear Audit** button, above the page.

See Also

[Audit Trail tab](#)¹⁰

3.3.4 Viewing the web output buffer

Always available for *Janus Debugger* sessions, the **Web Buffer** tab displays a program's web output buffer lines: the lines the Web Server application is preparing (in CCATEMP) to send to the browser at the completion of the request. Since the page is updated in real time as you step through a code program and PRINT and HTML statements are executed, you can view the **Web Buffer** page to watch your output HTML being built.

The page is cleared after the current request output is sent and the next web request is ready for execution.

The **Web Buffer** tab displays printable characters only. Binary and non-text content is not represented. Line-end characters *are* printable characters, though they may be represented by blanks.

The **Web Buffer** tab is also available when the *Sirius Debugger* is used to [debug web threads](#)¹⁵⁷.

See Also

[Controlling the execution of program code](#)⁵⁴

3.4 Tracing program execution

When debugging code, you may not want to step one statement at a time, nor to set a

breakpoint on a line and run until you hit it. You may want to run the program without breaking, collecting data (such as the lines that were executed and the value of a variable) as it changes. To do so, the Debugger provides several options for tracing execution. In all cases, the results of tracing are displayed in the **Execution Trace** tab.

To get information on any line in the execution trace, you double-click it, or right-click it and select **Procedure Information** from the context menu (described further in [Locating and editing procedures](#)^[126]).

If you invoke one of these tracing options after you have begun to debug a program, the tracing starts from the current execution point in the program. Statements executed prior to this point are not recorded. If you want to determine how you got to the current point (especially if complicated logic or several layers of calls were involved), the Debugger also provides an option for examining a history of the statements that already executed.

These subsections follow:

Tracing all lines executed^[130]

Trace all updates to a variable's value^[132]

Trace until a value change or until a value match^[133]

Displaying a statement history^[134]

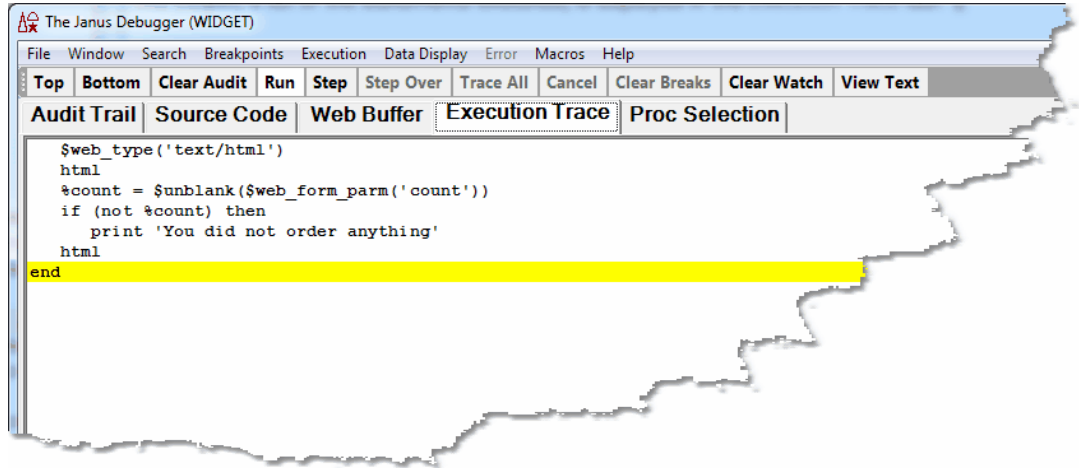
3.4.1 Tracing all lines executed

The simplest form of tracing starts from the current execution point and notes which lines were executed. You invoke this feature by clicking the **Trace All** button on the [button bar](#)^[39] (or by pressing the Ctrl+T keyboard combination (by [default](#)^[291]), or by selecting the **Trace To End** option in the Client's **Execution** menu).

The tracing continues until one of the following events:

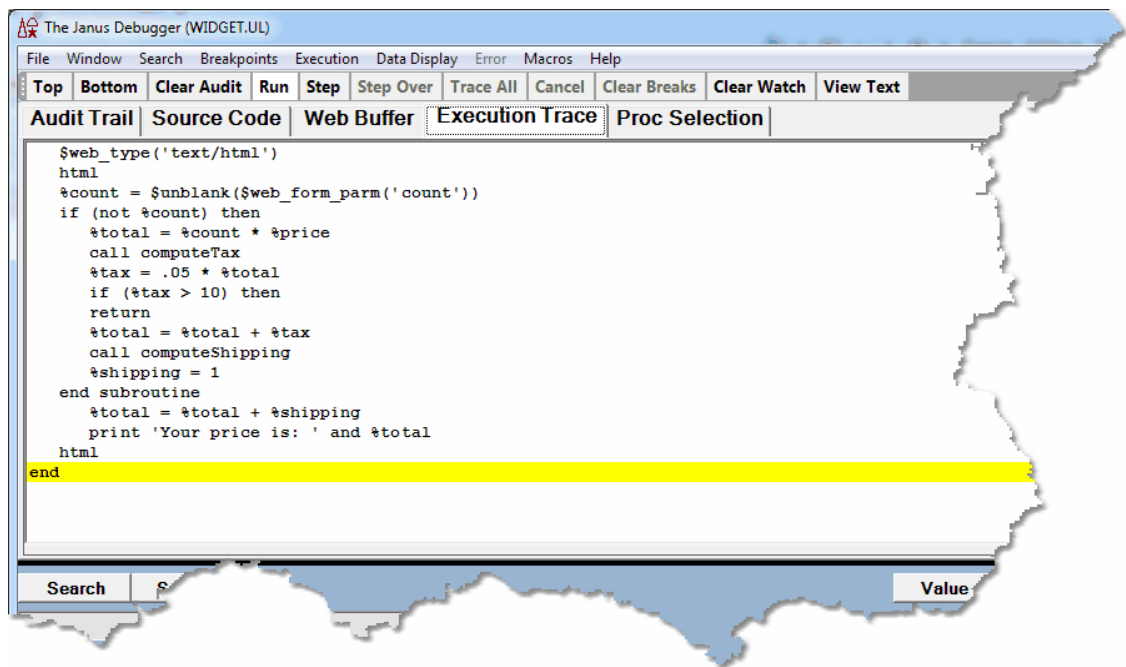
- The request ends
- A breakpoint is hit
- A cancelling error occurs
- A daemon is entered

The output, a list of the statements executed, is displayed in the **Execution Trace** tab:



If tracing was interrupted by a breakpoint or by daemon code, you can continue tracing by clicking the **Trace All** button again or by pressing the Enter key.

The **Execution Trace** tab display can help you with testing. You can see from the code in the example trace, above, that the execution path hit only the case where no orders are made. Contrast this with a trace of the same program where something was ordered:



3.4.2 Tracing all updates to a variable's value

Another type of tracing is to note all statements that modify a selected variable or data item and what value was assigned to the variable or item. To do this:

1. Enter a variable or data item name in the Entity-name input box below the main window.

Note: Items like global variables or *Model 204* fields or parameters require the additional prefixes before their names that are described in [Viewing and modifying program elements](#).^[87]

2. Click the Trace button:

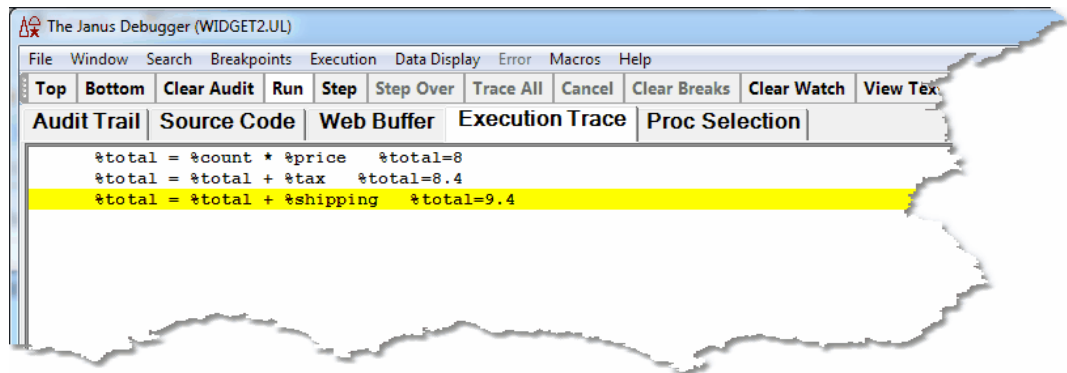


The `traceValues mappable`^[285] command and the **Trace Values** option in the Client's **Execution** menu have the same effect as the Trace button.

The trace continues until one of the events listed in [Tracing all lines executed](#)^[130] occurs.

3. View the result in the **Execution Trace** page.

Only lines that modified the variable are shown, along with the new value:



4. If tracing was interrupted by a breakpoint or by daemon code, you can continue tracing by clicking the Trace button again or by pressing the Enter key.

3.4.3 Tracing until a value change or until a value match

Another tracing option lets you step through a program, stopping on each statement that changes a specified data item. In this case, both the statement that modified the item and the resulting value are shown in the **Execution Trace** tab.

To trace until a variable value changes:

1. Enter a variable or data item name in the [Entity-name input box](#)^[51] below the main window.

Note: Items like global variables or *Model 204* fields or parameters require the additional prefixes before their names that are described in [Viewing and modifying program elements](#).^[87]

2. Click the **Run to Change** button:



The request runs until the specified item is modified.

If the value of the item does not change, execution runs until the end of the request or one of the events listed in ["Tracing all lines executed."](#)^[130]

3. View the result in the **Execution Trace** page.

If the request contains further changes to the value, you can click the **Run to Change** button again, or you can press the Enter key (which repeats the **Run to Change** action whenever the **Run to Change** button is highlighted).

The [runUntilVariableChanges](#)^[245] mappable command and the **Run Until Variable Changes** option in the Client's **Execution** menu have the same effect as the **Run To Change** button.

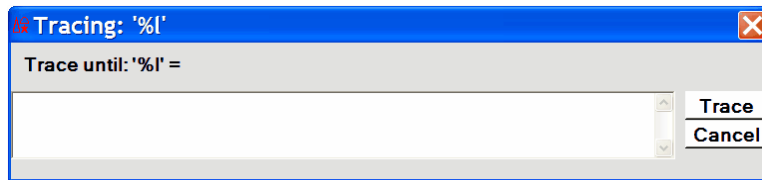
A variation of the "tracing until" technique lets you continue a request's execution until a selected item's value is equal to a value you specify. For example, you may want to use this feature to verify that a variable does not ever become a certain value.

To trace until a particular data item value:

1. Enter the item name (prefixed if necessary) in the Entity-name input box.
2. Press the Alt key while clicking the **Run to Change** button.

The [traceUntilVariableEqualsValue](#)^[275] mappable command and the **Trace Until Variable Equals Value** option in the Client's **Execution** menu have the same effect as Alt + the **Run To Change** button.

3. In the **Tracing** dialog box, specify the value of the variable at which tracing will stop:



4. Click the **Trace** button, at the right.

The request will run until the statement that makes the value of the selected item equal to the value you just specified, or the request will run until the end of the request or one of the events listed in ["Tracing all lines executed"](#)^[130].

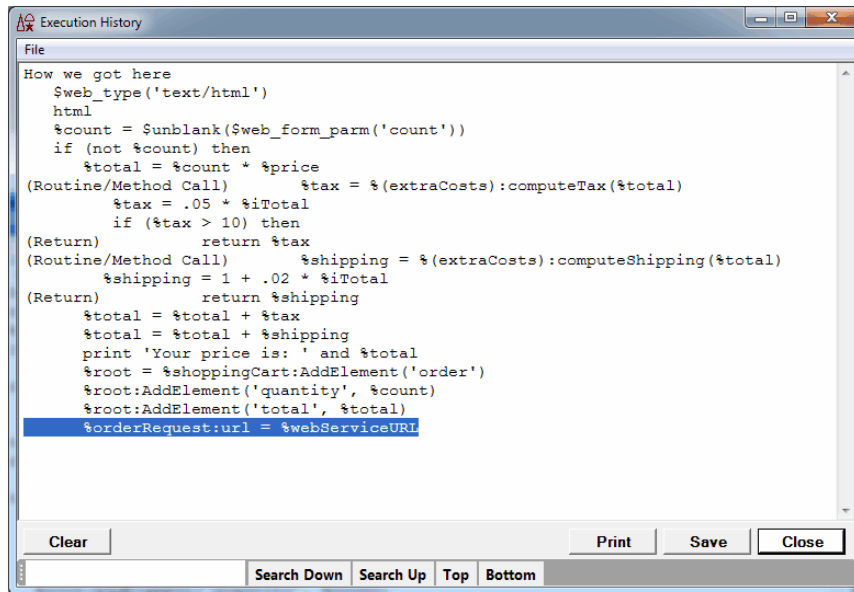
3.4.4 Displaying a statement history

If you are in the middle of debugging a lengthy or complicated program, and you have not invoked tracing, you can produce a display of all the statements that have *already* executed. Perhaps you got an unexpected runtime error or hit a breakpoint you set, and you want to review how you got to this point in the program.

To display a history of statement execution:

1. In the Client's **Execution** menu, select the **Get/Display History** option.

The **Execution History** window displays a history of statements executed up to the program's current execution point:



The execution history is as many as the last 1000 statements executed. Calls and returns for methods and subroutines are labeled:

- For a call, a **(Routine/Method Call)** indicator precedes the User Language statement.
- For a return, a **(Return)** indicator precedes the User Language statement.

While the **Execution History** window is open, you can interact with the main Client window, and any hot keys defined for the Client will work when the history viewer has focus.

2. Use the **Clear**, **Print**, **Save**, and **Close** buttons as necessary. And the search bar on the bottom of the window provides controls for searching the history.

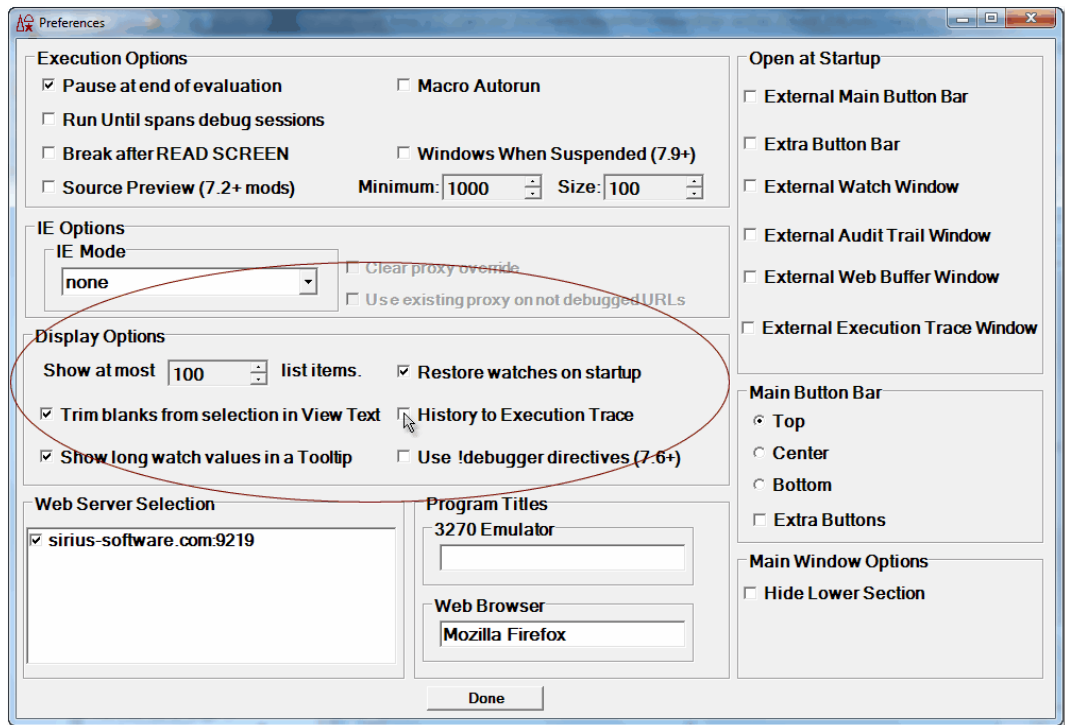
You can also invoke this feature with the [getHistory](#)^[211] mappable command, clear the window with the [clearHistory](#)^[190] command, and close the window with the [closeHistory](#)^[195] command.

Using the Execution Trace tab

You can arrange to view the statement history in the **Execution Trace** page instead of in the **Execution History** window:

1. Select **Preferences** from the **File** menu.

2. In the **Display Options** section of the **Preferences** dialog box, enable the feature by selecting the **History to Execution Trace** checkbox (it is clear by default), then click **Done**.



Note: The Client [setPreference](#)^[260] command has an option that lets you toggle the **History to Execution Trace** checkbox.

3. Invoke a statement history as usual from the **Execution** menu.

Using the Source Code tab

You can review the statements in the execution history in their actual context, that is, highlighted in the program in the **Source Code** tab (or **Daemon** tab). Four **Execution** menu options (or corresponding Client commands) let you select which executed statement(s) to view: the first or the last (that is, the statement at the beginning or the end of the history), or the previous or the next executed statement, relative to the currently highlighted line.

To inspect *in the program code* a previously executed statement:

1. At any point during the debugging of a request after some statements have been executed, either:
 - a. Open the **Execution** menu, and click one of the history-selection options: [Select Previous History Line](#)^[30], [Select Next History Line](#)^[30], [Select First History Line](#)^[30], [Select Last History Line](#)^[31].

Or:

- b. Run (mapped to a button or key, or via macro or command line) one of the Client commands that corresponds to the above-mentioned menu options: [previousHistory](#)^[237], [nextHistory](#)^[228], [firstHistory](#)^[208], [lastHistory](#)^[220].

You do **not** have to use the **Get/Display History** option to open the normal **Execution History** window, as described earlier in this section.

The Client responds by moving the current line to the executed statement you selected — in the **Source Code** or **Daemon** page that contains the statement.

2. Continue reviewing the executed statements by using the **Execution** menu history options or their command counterparts.

The Client locates and highlights the statement you select, and it removes the highlighting from the statement you viewed in the previous step.

Note: Although each history statement you view gets highlighted as if it were the current execution position, the actual execution position, the statement to be executed next if you invoke a debugging Step or Run operation, remains (non-highlighted) where it was when you began reviewing the statement history.

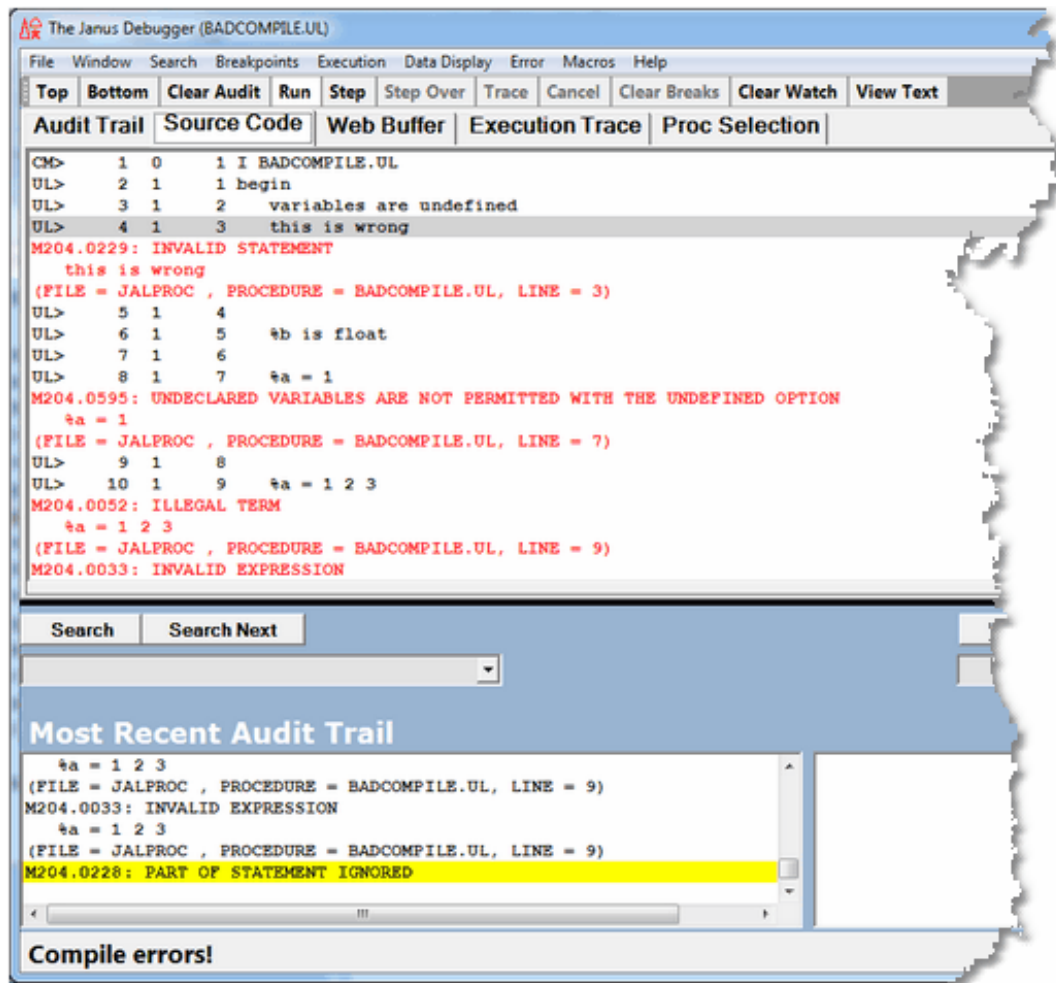
3. Resume normal debugging operations.

3.5 Viewing programs that contain coding errors

Debugger Client handling of programming errors differs according to the type of error: compilation errors or request-cancelling errors.

Compilation errors

If a program contains a compilation error, you can still use the Debugger to view the source code, along with the *Model 204* or Sirius error messages embedded (highlighted and prefixed with **ER**>) after the program statements that caused them:



As shown above, the Client status strip displays a **Compile errors!** message, and the **Run** button is the only program execution operation available.

- You can move to the next program statement that did not compile, if any, by pressing the F11 key (by [default](#)^[291]), by selecting the **Error > Next Compile Error** menu item, or by using a button to which you have [mapped](#)^[290] the [nextCompileError](#)^[228] command.

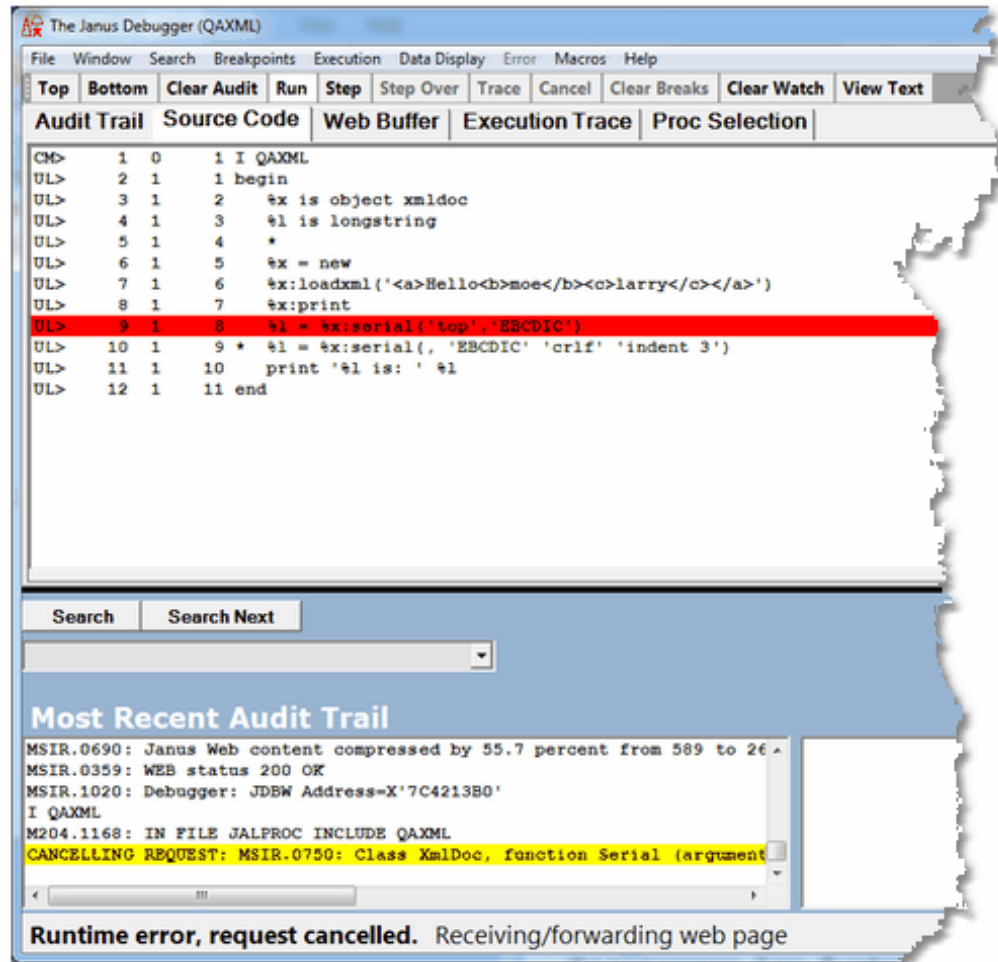
- You can move to the previous statement that did not compile, if any, by pressing the F10 key, by selecting the **Error > Previous Compile Error** menu item, or by using a button to which you have mapped the [previousCompileError](#)^[237] command.

Clicking **Run** (or selecting **Error > Quit**) invokes no further operation on the code, and (if *Janus Debugger*) the Web Server sends the compilation error messages to the browser, then advances to the next program, if any. If under the *Sirius Debugger*, the compilation error messages are sent to the terminal.

Request cancellation errors

If the program you are debugging contains a request-cancelling error, the Debugger lets you step through the program until you execute the statement that causes the error, or if you clicked **Run**, executes until the point of the error. At this point, the Client:

- Displays the source code (highlighting the line that has the error)
- Reports the values of any watched variables at the point of the error
- Reports the cancelling error message in the Client's audit trail displays
- Lets you inspect in the **Web Buffer** the results of statements that executed successfully prior to the error, if debugging a web request.



Furthermore, except in the case described below, the Client does not let you continue stepping through the request (Step buttons become unavailable). If you now click the Run button:

- If the *Janus Debugger*, the Debugger sends the cancelling error message to the **Web Buffer**, and it also sends an "internal server error" message to the browser.
- If the *Sirius Debugger*, the Client sends to the terminal the the cancelling error message as well as the results of any statements you executed successfully before the error.

An exceptional case is a cancelling error of the following type, which occurs within a User Language ON UNIT:

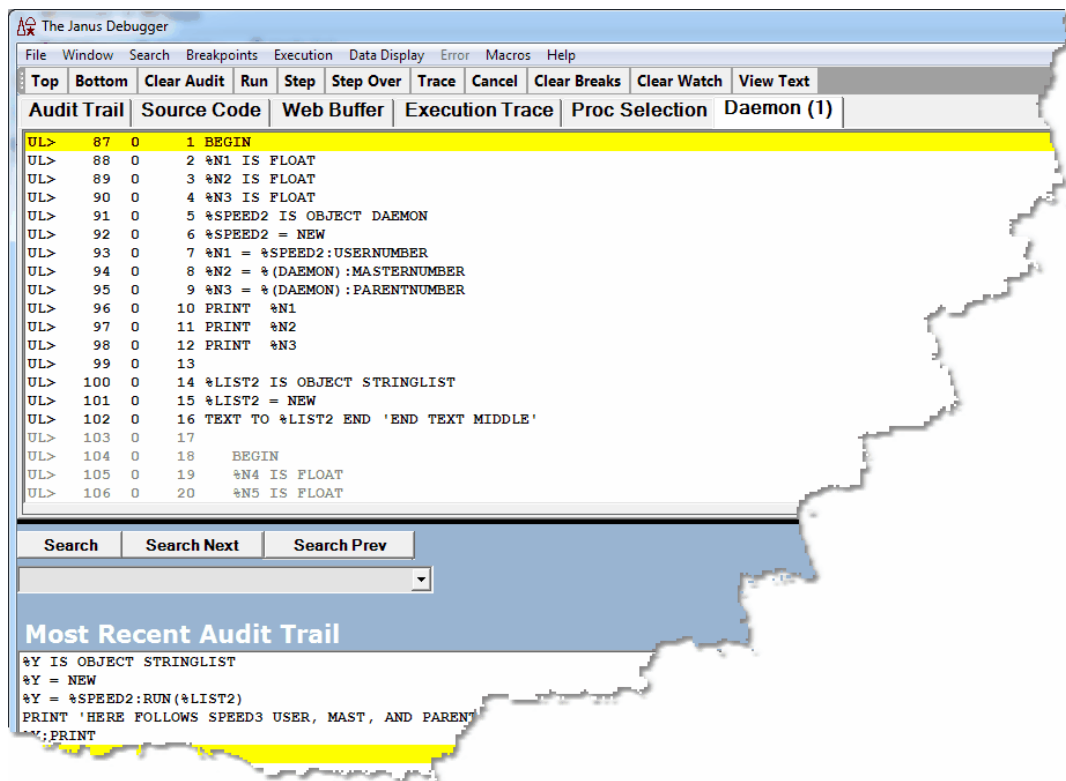
M204.1982: ILLEGAL JUMP ATTEMPTED OUT OF COMPLEX SUBROUTINE ON UNIT

If this error occurs, debugging is allowed to continue: you are *not* prevented from continuing to step through the request. This exception is designed for the debugging of ON UNIT code.

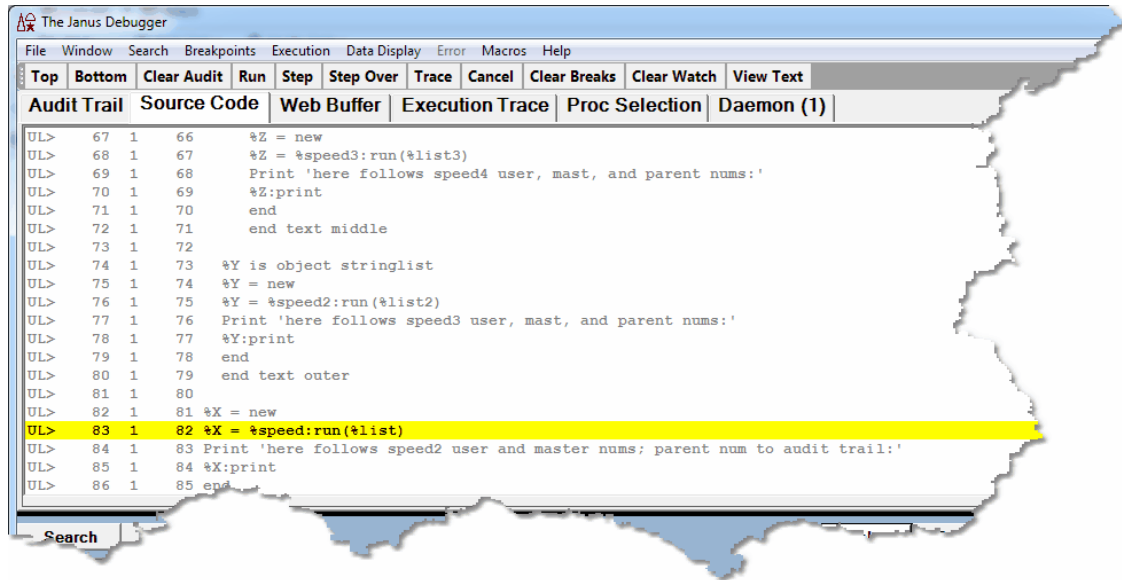
3.6 Debugging requests that spawn daemons

If the application you are debugging spawns a Sirius sdaemon in either of the following cases, the sdaemon code is displayed in a dynamically created **Daemon** tab:

- A *Janus SOAP* Daemon object **Run** method for an sdaemon or transactional sdaemon (shares record sets with the spawning thread, maintaining one logical unit of work)
- A synchronous \$COMMBG request (an output \$list parameter is specified for the function)



While the code in the **Daemon (1)** tab has focus, the current program in the **Source Code** tab waits for the sdaemon code to execute. The code line that spawned the sdaemon is highlighted, while the remaining **Source Code** lines are dimmed as a reminder that this code is not executable (the **Run**, **Step**, and **Step Over** buttons apply to the active code in the **Daemon (1)** tab). The **Source Code** lines are scrollable, however.



Commands you invoke, like **Step** or **Run**, execute for the code on the **Daemon (1)** tab. When the sdaemon execution is complete, the spawning program regains focus and the **Daemon (1)** tab is removed.

Multiple sdaemons

If your program has sdaemons that call additional sdaemons, the Debugger will display their code on a separate tab, dynamically spawning more Daemon tabs (**Daemon (2)**, **Daemon (3)**, and so on up to 5). Each new Daemon page will get focus and respond to the code execution buttons, while the **Source Code** page and preceding Daemon pages will wait for the current sdaemon execution to complete. After the current sdaemon code runs to completion, its tab will be removed, **Daemon thread completed** will be displayed in the Status bar, and the preceding Daemon page will regain focus.

Discontinue Daemon debugging

You can optionally have the Debugger run in a mode that does not display sdaemon code in the Debugger Client. In this mode, the Debugger runs all other code [as usual](#)^[56] but suspends the display (but not the execution) of sdaemon code.

To enter this mode, from the **Source Code** page at any point during program execution, you simply press the Alt key and click the **Run** button (the Alt+F5 key combination is also equivalent, by [default](#)^[291]). Equivalent alternatives are the [runWithoutDaemons](#)^[246] command and the **Run Without Daemons** option of the **Execution** menu.

Once this mode is entered, all debugging of Daemons is suspended:

- Subsequent sdaemon code executes normally but is not displayed

- No **Daemon (1)** tab is shown in the GUI
- No sdaemon source code is sent from the mainframe to the PC

As in normal running mode, this "discontinue-Daemon-debugging" mode stops on breakpoints. For example, if you have a program that spawns five daemons, only the last of which you want to debug, you could:

1. [Put a breakpoint](#)^[58] on the line that invokes the fifth daemon (or just before it).
2. Simultaneously press the Alt key and click the Run button.
3. Click the Run button.

The Debugger Client will skip over the first four daemons but stop on your breakpoint.

Debugging an interactive daemon

An sdaemon you have spawned with a *Janus SOAP* Daemon class **Run** method (or **RunAsynchronously** or **RunIndependently**) might temporarily return control to its master thread by issuing a **ReturnToMaster** method. The master thread may do further processing and then return control to the sdaemon by issuing a **Continue** method (or **ContinueAsync** or **ContinueIndependently**). There may be multiple such exchanges of control.

This interactive processing is captured in the Debugger by:

1. Keeping a tab (**Source Code**) open for the code of the master thread and a tab (**Daemon**) open for the sdaemon, as usual.
2. Giving focus to the tab whose code is executing, and dimming the code in the other tab.

If a master thread issues a **ContinueAsync** or **ContinueIndependently** call, however, the focus does not shift to the **Daemon** tab (though its code remains available for inspection).

See Also

[Controlling the execution of program code](#)^[54]

[Source Code tab](#)^[11]

3.7 Debugging Web Server persistent sessions

You can use the *Janus Debugger* to debug *Janus Web Server* "persistent session" applications. These are applications in which Online program execution is suspended while a browser user returns data to the program, whereupon the program continues. The Debugger supports two such types of Web Server programs:

- HTML form processing using the `$WEB_FORM_DONE` function
- *Janus Web Legacy Support*, which processes User Language 3270-screens in a web browser

`$WEB_FORM_DONE` sessions

Using the `$WEB_FORM_DONE` function, a User Language procedure can serve an HTML form, suspend execution until the form is submitted, then resume executing the program at the point where it was suspended.

To debug a `$WEB_FORM_DONE` application:

1. Invoke the program from your browser; then, in the Debugger Client, step through the program as usual.
2. When a `$WEB_FORM_DONE` executes, the execution of the User Language program pauses, and **Session awaits browser** displays in the [Status bar](#)^[50].
3. Move from the Debugger Client to your web browser, respond to the directives in the HTML form, and submit the form.
4. Return to the Debugger Client, where **Persistent Session Resumed** displays in the Status bar, and continue debugging.

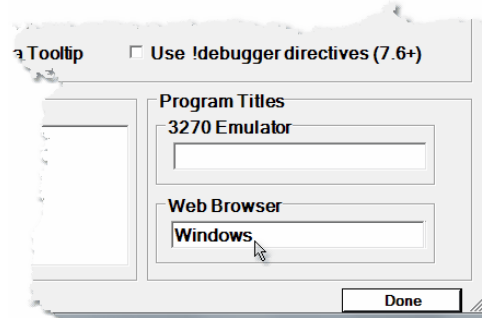
Note: As a convenience, the Debugger can make your web browser window the topmost on your PC screen when the Client pauses as it processes the `$WEB_FORM_DONE` call.

To invoke this feature:

1. In the Client, select **Preferences** from the **File** menu (or use the Ctrl+P keyboard shortcut).

2. In the **Preferences** dialog box, locate the **Web Browser** text box (in the **Program Titles** section) and provide a text string that matches some or all of the browser-identifying title that displays at the top of the browser window.

The characters in your matching string can be any case and match anywhere in the browser title. Any trailing blanks you enter are preserved.

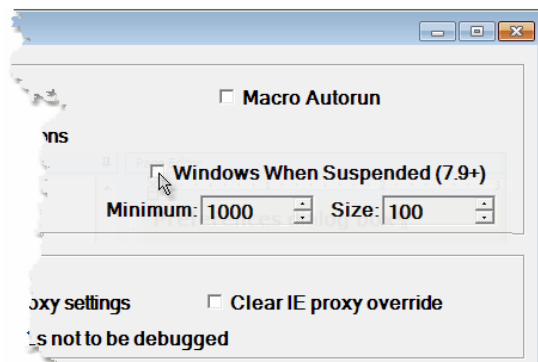


3. Click **Done**.

The feature takes effect at the next execution of a \$WEB_FORM_DONE statement. The **Web Browser** setting that exists at the end of the Debugger Client session persists to the next run of the Client.

Note: By default, this feature does not take effect if the Client is not executing the part of a program that contains the \$WEB_FORM_DONE. For example, the statement might be in code selected to be [excluded from debugging](#).^[66]

However, if you also select the **Windows When Suspended** option in the **Preferences** dialog box (**Execution Options** section), you can make the feature apply whenever this statement occurs, even in code the Debugger is not actively executing.



Legacy Support sessions

By automatically converting 3270 screens to HTML, the *Janus Web Legacy Support* feature lets you run under the Janus Web Server applications that do 3270 full screen reads. You can debug such an application with the *Janus Debugger* only if you also license the *Sirius Debugger* and you are running under Version 6.9 of the *Sirius Mods* or higher.

To debug a *Legacy Support* application:

1. Invoke the User Language program from your browser, and in the Debugger Client, step through the program as usual.
2. When a READ SCREEN executes, the execution of the program pauses, and **Session awaits browser** displays in the [Status bar](#)⁵⁰.

If you have not also purchased the *Sirius Debugger*, the web thread is softly restarted and the debugging session is terminated.
3. Move from the Debugger Client to your web browser, respond to the screen (now HTML form) prompts, and submit the form by clicking the Enter button or a PF key button.
4. Return to the Debugger Client, where **Persistent Session Resumed** displays in the Status bar, and continue debugging.

These events are reported in a sequence of lines in the Client **Audit Trail** page like the following:

```
2010 11 08 10:01:48.27  2  20 LI I SCREENO
2010 11 08 10:01:48.27  2  20 MS M204.1168: IN FILE GWDEB INCLUDE SCREENO
2010 11 08 10:02:10    Full Screen Read Pending
2010 11 08 10:02:22    READ SCREEN completed
```

Note: As described above for \$WEB_FORM_DONE applications, the Debugger can bring your browser to the top on your PC screen when the Client pauses for the READ SCREEN.

To invoke this feature:

1. Select **Preferences** from the **File** menu.
2. In the **Preferences** dialog box, locate the **Web Browser** text box (in the **Program Titles** section) and provide a text string that matches some or all of the title that displays at the top of the browser window.

The characters in your matching string can be any case and match anywhere in the title. Any trailing blanks you enter are preserved.

3. Click **Done**.

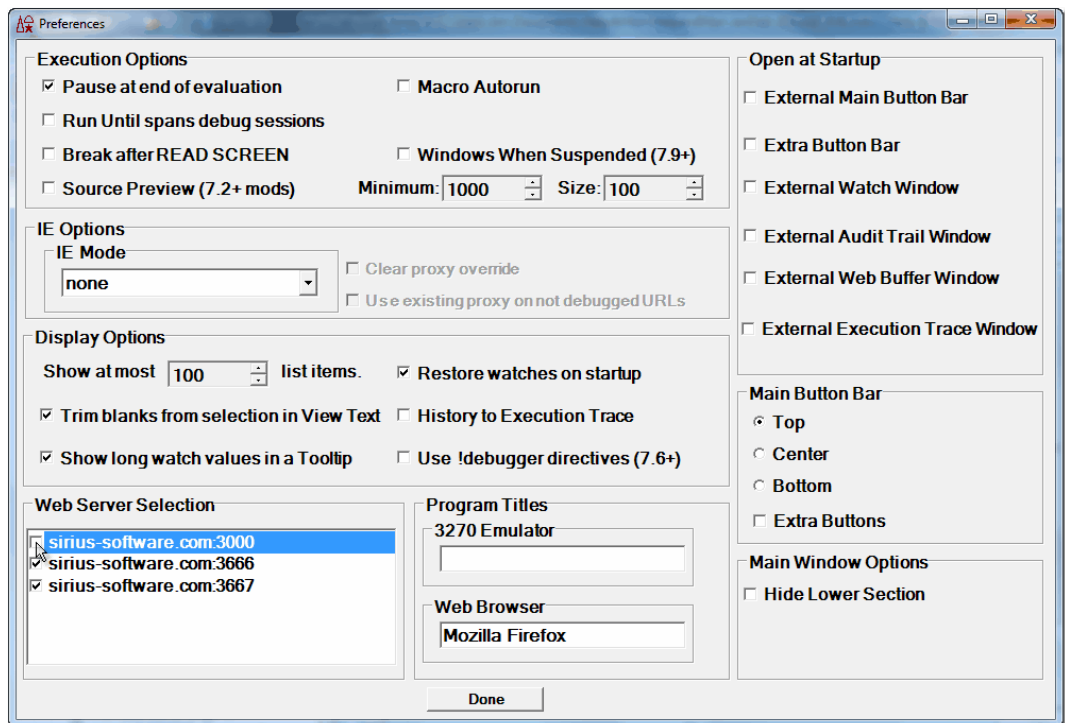
3.8 Debugging multiple Web Servers

You can configure the *Janus Debugger* to debug applications that run on any of multiple Janus Web Servers. By default, a browser request sent to any of these web servers is intercepted by the Debugger Client proxy for the purpose of debugging. If you want to temporarily turn off debugging of one or more of these web servers, you can do so dynamically in the Debugger Client.

You configure the *Janus Debugger* to debug any of multiple Janus Web Servers by manually editing the `debuggerConfig.xml` file, as [described](#)^[374] in the product installation information.

To disable the debugging of web requests for a particular Web Server:

1. In the Client, select **Preferences** from the **File** menu (or use the Ctrl+P keyboard shortcut).
2. In the **Preferences** dialog box, in the **Web Server Selection** list, clear the checkbox next to the entry for the web server you want to disable, then click **Done**.



Browser requests for this Web Server will no longer be debugged. This setting takes effect immediately, and it persists over multiple runs of the Debugger Client.

Disabling the debugging of a server does not remove it from the `debuggerConfig.xml` file. You can re-enable the debugging of a server by marking its **Web Server Selection** checkbox.

CHAPTER 4 *Additional Debugger Functionality*

These sections are included:

[Copying, printing, or saving text^{\[149\]}](#)

[Using the SIRIUS DEBUG command^{\[151\]}](#)

[Using the SIRIUS DEBUG command for web threads^{\[157\]}](#)

[Debugging SSL applications^{\[159\]}](#)

[Debugging Web Service applications^{\[160\]}](#)

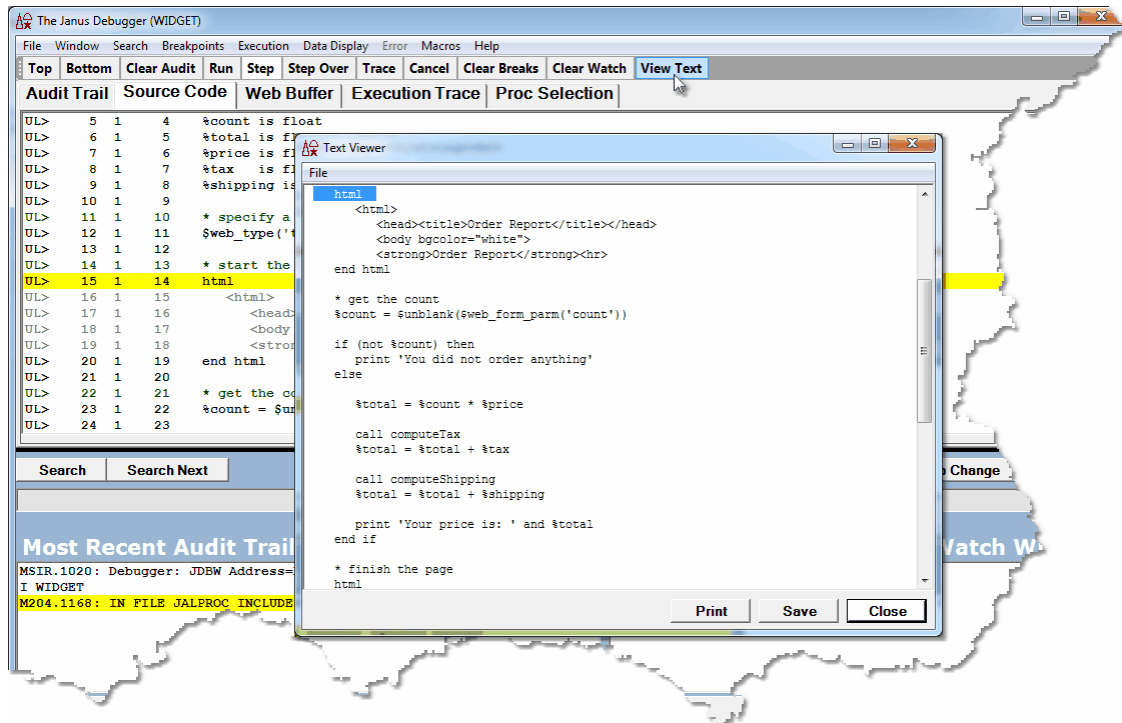
[Using the DebuggerTools class methods^{\[161\]}](#)

[Using a local editor^{\[166\]}](#)

4.1 Copying, printing, or saving text

You can invoke a separate text viewer window in which you can display, copy, print, and save the text data contained in a Client tabbed page.

The viewer copies the contents of the current Client tab when you click the button or press the hot key to which you have assigned the `viewText` command (as described below):



The scrollable, expandable **Text Viewer** window displays just the text content of the currently active tab (excluding, for example, the additional line numbers and identifying information that the Client automatically adds to the **Source Code** page display). The viewer scrolls to and highlights the current line from the page that is copied. As many as one thousand lines of the tabbed-page text, starting from the top, are copied.

Other features include:

- The viewer works with lines from any Client tab except the **Proc Selection** tab.
- The viewer has **Print** and **Save** buttons, which act on the entire copied content, whether currently visible or not.
- Familiar Windows **Print**, **Page Setup**, and **Print Preview** dialog boxes are accessible from the viewer's **File** menu.
- Standard mouse-based copying of text data is enabled, but altering or deleting text is **not** allowed.
- You can automatically remove any leading and trailing blanks from selections you copy.

To enable the **Text Viewer** window:

1. Create a text file named `ui.xml` in the folder where the `JanusDebugger.exe` file is installed, or edit the existing `ui.xml` file.
2. In the `ui.xml` file, map the `viewText`^[281] command to a Client button or keyboard shortcut, as described in ["Setting up the ui.xml file"](#)^[287].

For example, this mapping associates a `viewText` command with `button5`:

```
<mappings useDefaults="true">
  <mapping command="viewText" button="button5"/>
</mappings>
```

3. Save the `ui.xml` file, and [restart](#)^[18] the Debugger Client.
4. If you want to enable automatic trimming of leading and trailing blanks from data you copy to the Text Viewer:
 - a. Select **Preferences** from the Client **File** menu.
 - b. In the **Display Options** area, mark the **Trim blanks from selection in View Text** checkbox.
 - c. Restart the Debugger Client.
5. Select the Client tab whose text you want to copy, and click the **View Text** button, press the key you assigned to the `viewText` command, or select the **View Text** option of the **Window** menu.

4.2 Using the SIRIUS DEBUG command

Unlike the *Janus Debugger*, the *Sirius Debugger* is started and stopped by a command switch, the SIRIUS DEBUG command. The command, which can be issued from the *Model 204* command prompt or in a procedure, has these mutually exclusive subcommands:

```
SIRIUS DEBUG { [ON|OFF] [SUSPEND|RESUME] [CLIENTCOMMAND] [STATUS] }
```

As described below:

- The [ON](#)^[152] subcommand starts a *Sirius Debugger* session (it requires additional parameters). It can also be used for [debugging Janus Web threads](#)^[157].
- The [OFF](#)^[154] subcommand stops a debugging session.
- The [SUSPEND](#)^[155] and [RESUME](#)^[156] subcommands discontinue and continue a debugging session.

- The [CLIENTCOMMAND](#)^[155] subcommand sends a command to the Debugger Client.
- The [STATUS](#)^[156] subcommand gets a status report about the worker threads for the *Janus Debugger*, the *Sirius Debugger*, or both.

The SIRIUS DEBUG commands that change the Debugger's state from on to off or from suspended to resumed may interchangeably be issued from the command line or within a procedure. You can control the Debugger's state entirely from the command line, entirely through one or more procedures, or from a combination of both places.

SIRIUS DEBUG commands issued from a procedure must *not* be placed between explicit procedure BEGIN and END statements.

SIRIUS DEBUG ON

To initiate a *Sirius Debugger* session, you issue the SIRIUS DEBUG ON command from the *Model 204* command prompt or within a BATCH2 input stream or procedure. The command requires the parameters described below that identify the network ports and the workstation used in your debugging session. The values of these parameters are established during product installation:

SIRIUS DEBUG ON [janClientPort] [pcHost] [pcPort] [workerPort][janClientPort]

where:

<i>janClientPort</i>	<p>The name of the Janus client socket port that is defined^[368] for the <i>Sirius Debugger</i> to use to contact the Debugger Client workstation.</p> <p>As of <i>Sirius Mods</i> version 7.9, the default value is an asterisk (*), which means to use a Janus port whose definition includes the MASTER parameter. Such a port can be accessed on a Debugger connection request without specifying its port name on the SIRIUS DEBUG ON command.</p> <p>This port must be started.</p>
<i>pcHost</i>	<p>The workstation running the Debugger Client. This may be an IP number or a DNS name, as described^[370] during product installation.</p> <p>As of <i>Sirius Mods</i> version 7.9, the default value is a period (.), which means to use the IP address of the machine from which the TN3270 session was initiated.</p>
<i>pcPort</i>	<p>The workstation port number on which the Debugger Client is listening. As described^[370] during product installation, this is typically 8081.</p>

The *Model 204* User 0 parameter SDEBGUIP sets the default value of this port. Valid values for this parameter, which is also resettable by the system manager, range from 0 to 65535. If 0, no default is set, and the parameter's value must be explicitly specified in the SIRIUS DEBUG ON command. A **VIEW SDEBGUIP** command returns the current setting of SDEBGUIP.

Note: As shown in an example below, to omit an explicit specification of *pcPort* (letting it default to the setting of SDEBGUIP) but still provide an explicit value for *workerPort*, use an asterisk (*) for the missing *pcPort* specification to indicate the respective position of the parameter values.

workerPort

The port number in your Online that [is defined](#)^[367] for worker threads. This can be the same port number that provides worker threads for the *Janus Debugger*, as well.

The *Model 204* User 0 parameter SDEBWRKP sets the default value of this port. Valid values for this parameter, which is also resettable by the system manager, range from 0 to 65535. If 0, no default is set, and the parameter's value must be explicitly specified in the SIRIUS DEBUG ON command. A **VIEW SDEBWRKP** command returns the current setting of SDEBWRKP.

For example:

SIRIUS DEBUG ON DEBCLIENT 198.242.244.235 8081 3226

After issuing the command, you receive a message similar to this:

***** MSIR.0915: Debugging is on; client is 198.242.244.235 port 8081, sessionID: 00000069D812279**

If you had set the workstation and worker port numbers with the SDEBGUIP and SDEBWRKP User 0 parameters, as revealed by a **VIEW** command:

VIEW SDEBGUIP,SDEBWRKP

**SDEBGUIP 8081 SIRIUS DEBUG DEFAULT GUI PORT NUMBER
SDEBWRKP 3226 SIRIUS DEBUG DEFAULT WORKER PORT NUMBER**

The previous **DEBUG ON** command could be simplified:

SIRIUS DEBUG ON DEBCLIENT 198.242.244.235

In this case, you receive different confirmation messages:

***** MSIR.0942: Sirius Debugger GUI port defaulted to 8081
*** MSIR.0942: Sirius Debugger Worker port defaulted to 3226
*** MSIR.0915: Debugging is on: client is 198.242.244.235 port 8081, sessionID: sessionID: 00000069D812279**

You can also use asterisks to indicate that the port defaults are to be taken. These two commands are equivalent:

```
SIRIUS DEBUG ON DEBCLIENT 198.242.244.235 * *
```

```
SIRIUS DEBUG ON DEBCLIENT 198.242.244.235
```

You receive an error message if you use either of the above commands without having set the default port values. To take the GUI port default but specify a value for the worker port, you must use an asterisk in the GUI port position:

```
SIRIUS DEBUG ON DEBCLIENT 198.242.244.235 * 3226
```

You can further simplify the connection command by taking advantage of the *janClientPort* and *pcHost* parameter defaults (added in *Sirius Mods 7.9*). If the DEBCLIENT port is defined with the MASTER parameter, and SDEBGUIP and SDEBWRKP are as above, these commands are equivalent:

```
SIRIUS DEBUG ON * .
```

```
SIRIUS DEBUG ON
```

On the Debugger Client, **Connection from Online** displays in the **Status bar**⁵⁰ after the DEBUG ON command is issued. This indicates that the Debugger is "on" and awaiting the next program to debug.

After the command runs successfully, any User Language program you initiate from the *Model 204* command line will appear in the **Source Code** tab of the Debugger Client GUI for debugging.

SIRIUS DEBUG OFF

SIRIUS DEBUG OFF stops a debugging session. The command is issued as is from the *Model 204* command prompt (or from your BATCH2 stream or a procedure); it has no additional parameters.

After issuing the command, you should receive this response in *Model 204*:

```
*** MSIR.0913: Sirius Debugger is now off
```

On the Debugger Client, **Online has disconnected** displays in the Status bar.

You can also turn off the *Sirius Debugger* by logging off of *Model 204* (any logoff is an implied SIRIUS DEBUG OFF).

Note: Explicitly turning off the Debugger is necessary if you are using the *Janus Debugger* as well as the *Sirius Debugger* for the same Online and worker port. To switch from a *Sirius Debugger* session to a *Janus Debugger* session, you must explicitly drop the *Sirius Debugger* session. The *Janus Debugger* automatically closes its connections and does *not* require an explicit notification to switch or end a session.

The `turnOffDebugging` [mappable command](#)^[285] for a Client button, hot key, or macro is equivalent to SIRIUS DEBUG OFF. This alternative lets you turn off the Debugger at any time during the debugging of a program, and it provides even more flexibility than using a procedure to issue SIRIUS DEBUG OFF.

SIRIUS DEBUG SUSPEND

SIRIUS DEBUG SUSPEND immediately discontinues the debugging of the current request, but preserves the connection from the Online thread to the Debugger Client (that is, debugging is inactive but in a state where it can be readily reactivated). Debugging can continue if a SIRIUS DEBUG RESUME command is issued from the command line or in an included procedure.

Issuing SIRIUS DEBUG SUSPEND has no effect and receives a harmless error message in either of these cases:

- Debugging is not currently active for this thread (via SIRIUS DEBUG ON or SIRIUS DEBUG RESUME).
- Debugging is already suspended (via SIRIUS DEBUG SUSPEND).

Once you suspend debugging, the current program (if any) executes immediately, and the Debugger continues in the suspended state until you issue a SIRIUS DEBUG RESUME or a SIRIUS DEBUG OFF command. While debugging is suspended, you can execute *Model 204* commands from the command line as usual; the difference between debugging being suspended and being off is that the Client remains in a "waiting" state.

Note: SIRIUS DEBUG OFF turns off debugging for a thread for which debugging is suspended, but SIRIUS DEBUG ON does **not** resume debugging a thread for which debugging is suspended.

SIRIUS DEBUG CLIENTCOMMAND

Available as of version 7.8 of the *Sirius Mods*, SIRIUS DEBUG CLIENTCOMMAND lets you send a [command](#)^[179] or [macro](#)^[309] to be invoked by the Debugger Client. The single parameter of SIRIUS DEBUG CLIENTCOMMAND is the command or macro you want to send, specified without regard for case:

SIRIUS DEBUG CLIENTCOMMAND [COMMAND | MACRO] *command*

If *command* is preceded by a keyword or followed by its parameter(s), such a "clause" must be quoted. See the examples below.

If *command* is not qualified by the keyword **Command** or **Macro**, the Debugger Client searches first for a macro named *command*, then for a Client command named *command*. You can use the **Command** or **Macro** keyword to search exclusively for a specified command or exclusively for a specific macro. The macro search is restricted to the Client installation folder or the [designated](#)^[296] macro folder.

Three examples follow:

```
SIRIUS DEBUG CLIENTCOMMAND clearWatch
```

```
SIRIUS DEBUG CLIENTCOMMAND 'Addwatch %watchthis'
```

```
SIRIUS DEBUG CLIENTCOMMAND 'Macro mymacro %s'
```

If the *Sirius Debugger* is not currently in a session, issuing SIRIUS DEBUG CLIENTCOMMAND has no effect and receives a harmless error message.

You can also execute Client commands and macros from the [ClientCommand](#)^[164] and [Command](#)^[164] methods of the DebuggerTools class.

SIRIUS DEBUG RESUME

SIRIUS DEBUG RESUME lets you resume debugging that was previously suspended with SIRIUS DEBUG SUSPEND. When you issue SIRIUS DEBUG RESUME (from the command line or in a procedure) while debugging is suspended, the Debugger immediately returns to normal debugging mode.

Issuing SIRIUS DEBUG RESUME has no effect and receives a harmless error message in either of these cases:

- Debugging is already active for this thread (via SIRIUS DEBUG ON or SIRIUS DEBUG RESUME).
- Debugging is currently off (via SIRIUS DEBUG OFF or because it has yet to be initiated for this thread via SIRIUS DEBUG ON).

You can use SIRIUS DEBUG OFF (or the [turnOffDebugging](#)^[276] mappable Client command) to turn off debugging for a thread for which debugging is resumed.

Note: After a successful SIRIUS DEBUG RESUME, the Client restores any [White List](#)^[79] or [Run Until](#)^[75] processing that was active prior to the SIRIUS DEBUG SUSPEND.

SIRIUS DEBUG STATUS

SIRIUS DEBUG STATUS provides a simple status report about the worker threads for *Sirius Debugger* sessions, *Janus Debugger* sessions, or both for a given *Model 204 Online*.

The command is issued as is from the *Model 204* command prompt (or at the end of your BATCH2 stream); it has no arguments. After issuing the command, you receive a display like the following:

```
*** Janus/Sirius Debugger status: Total workers=12
*** Janus/Sirius Debugger status: Janus Debugger Sessions=3
*** Janus/Sirius Debugger status: Sirius Debugger Sessions=1
*** Janus/Sirius Debugger status: Total Active Sessions=4
*** Janus/Sirius Debugger status: Draining=0
*** Janus/Sirius Debugger status: Available=8
*** Janus/Sirius Debugger status: Janus Debugger Session HWM=5
*** Janus/Sirius Debugger status: Sirius Debugger Session HWM=1
```

where:

Total workers	The number of workers created with the DEBUGMAX ^[366] User 0 parameter
Janus Debugger Sessions	Worker threads currently being used for <i>Janus Debugger</i> sessions, if any
Sirius Debugger Sessions	Worker threads currently being used for <i>Sirius Debugger</i> sessions
Total Active Sessions	<i>Janus Debugger</i> Sessions + <i>Sirius Debugger</i> Sessions
Draining	Workers that are transitioning from Active to Available
Available	Total workers - (Total Active Sessions + Draining)
Session HWM	The greatest value that the number of concurrent users has reached, per Debugger product (<i>Janus Debugger</i> and <i>Sirius Debugger</i>) since the Online was started.

4.3 Using the SIRIUS DEBUG command for web threads

If you license the *Janus Debugger*, you can use the SIRIUS DEBUG command to invoke the debugging of programs served by *Janus Web Server* threads. This may be useful if you need to avoid changing the proxy server settings on your web browser, for example.

You invoke a Janus Web procedure from your browser, a **SIRIUS DEBUG ON** command you embedded in the procedure starts the Debugger, and you work with your source code in the Debugger Client as usual. The thread you are debugging counts as one of your *Janus Debugger* [authorized "seats."](#)^[365]

Using the SIRIUS DEBUG command to invoke debugging requires no additional [configuration](#)^[363] of the Debugger. You must make sure, however, that no proxy server is defined for the browser with which you send a request to the *Janus Web Server*.

To debug a Janus Web thread:

1. In the Janus Web program you want to debug, insert a [SIRIUS DEBUG ON command](#)^[152] to invoke the *Janus Debugger*.

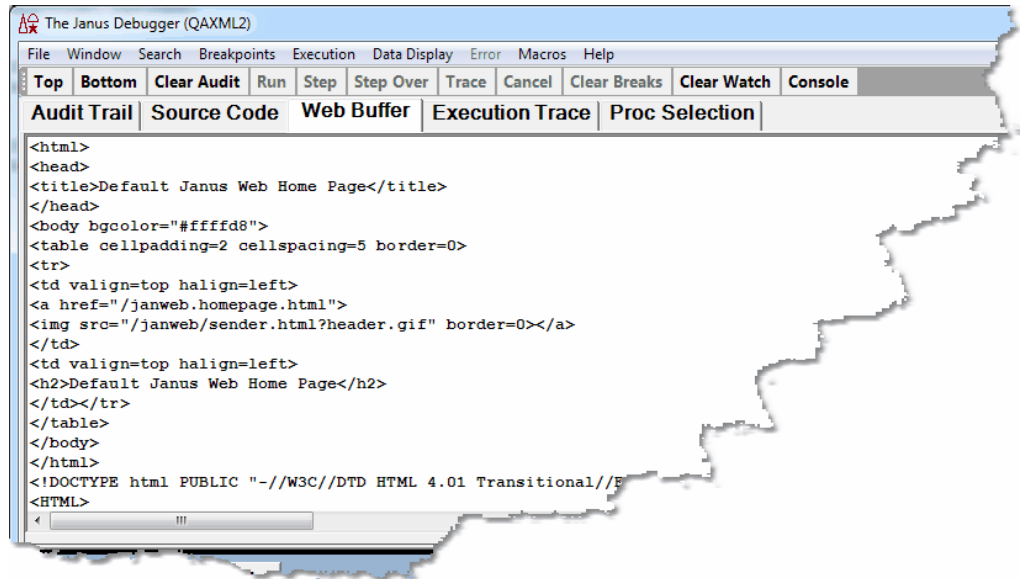
The command must **not** be placed between explicit BEGIN and END User Language statements.

2. Start the Debugger Client.
3. Make sure the Client is **not** automatically maintaining a proxy server for you:
 - a. From the **File** menu, select **Preferences**.
 - b. Make sure the **Automatically Maintain IE proxy settings** checkbox is clear.
4. In the browser you use to invoke debugging, confirm that the Debugger Client is not [defined as a proxy server](#)^[380].
5. From your browser, invoke the URL for the web program that contains the **SIRIUS DEBUG ON** command.

The browser pauses in a loading state, and the web program is sent to the Debugger Client **Source Code** page.

6. Work with the procedure code in the Client as usual.

As the code is evaluated, output [destined for a web page](#)^[129] goes to a Web Buffer page in the Client:



When code execution completes, output (including error messages and Web Buffer contents) is sent to the browser instead of to the *Model 204* "terminal." At this point, the connection to the web thread is closed and the debugging session ends. It is not necessary to provide an explicit **SIRIUS DEBUG OFF** command.

You can also embed **SIRIUS DEBUG SUSPEND and RESUME**^[155] commands in the procedures that are included in your program to take advantage of those features. And you can similarly embed a **SIRIUS DEBUG OFF** command or invoke its equivalent programmable command [turnOffDebugging](#)^[276] from a Client button or hot key.

Once debugging is on and the Client has the web thread source code, **SIRIUS DEBUG** command input from the command line at the *Model 204* host has no effect on debugging — the command line thread is separate from the web thread.

4.4 Debugging SSL applications

If you are using Janus Network Security, you may want to debug User Language programs that run on a secure Web Server (normally accessed with URLs beginning with "https").

The *Janus Debugger* manages SSL applications by maintaining an unencrypted connection between the web browser and Debugger Client, and a secure connection between the Debugger Client and the secure Web Server. This lets the Client examine the HTTP request and response in unencrypted form, while all data to and from the Web Server travels the network in encrypted form.

To use the Debugger with a secure Web Server, you must "inform" the Debugger Client that it must connect to the Web Server using the Secure Sockets Layer (SSL) protocol. You do this simply by adding a line (an empty `ssl` element) to the Client configuration XML file for each secure server you debug, as [described](#)^[374] in the installation documentation.

Then, in the URL to access the secure site, you use `http://` instead of `https://`.

A Short SSL Example

If you use <https://secureapp.myhost.com:123> to access your secure Janus Web application, do the following to debug this application with the *Janus Debugger*:

1. In the `serverList` element in the `debuggerConfig.xml` file (located in the same directory as `JanusDebugger.exe`), add a `server` element (as [described](#)^[374] in the installation instructions) that includes an empty `ssl` sub-element (which you can specify either as `<ssl/>` or as `<ssl></ssl>`):

```
<serverList>
  <server>
    <host>secureapp.myhost.com</host>
    <webPort>123</webPort>
    <workerPort>321</workerPort>
    <ssl></ssl>
  </server>
</serverList>
```

2. [Point](#)^[380] your web browser at the Debugger Client.
3. Start, or restart, the Debugger Client.
4. From your browser, access your application with this URL:

`http://secureapp.myhost.com:123`

Note: HTTPS is **not** specified in the URL.

4.5 Debugging Web Service applications

While the *Janus Debugger* is designed for debugging HTTP server applications whose client is a web browser (that is, *Janus Web Server* applications), the *Sirius Debugger* also lets you debug an HTTP server application whose client is a User Language HTTP socket program written with the *Janus Sockets* HTTP Helper. This Debugger versatility is possible because the Debugger Client functions as a proxy server, and the HTTP Helper can use a proxy server.

To debug a User Language Web Service client:

1. Install and configure the *Sirius Debugger* normally.

- Point the HTTP Helper application code to the Debugger Client.

Add code like the following to your HTTPRequest object code:

```
%HTTPReq is object HttpRequest auto new
%HTTPReq:URL = 'http://mywebservice
if (%(DebuggerTools):AmDebugging) then
    %HTTPReq:Proxy = http://workstation:port
end if
```

where:

- AmDebugging is a [DebuggerTools class](#)^[161] shared method that queries whether or not the program is being run under the Debugger.
- workstation is the IP address of your Debugger Client workstation, and port is the workstation listening port number you specified in the [Debugger configuration file](#)^[369].

One way to get the workstation IP number is to use the **IPConfig** command at the MS DOS prompt on your workstation. For example:

```
C:\Documents and Settings\username> ipconfig
```

The response will show your port number:

```
Windows IP Configuration
```

```
Ethernet adapter Wireless Network Connection:
```

```
Media State . . . . . : Media disconnected
```

```
Ethernet adapter Local Area Connection:
```

```
Connection-specific DNS Suffix  : hsd1.ma.comcast.net.
```

```
IP Address. . . . . : 192.168.1.104 <== you want this
```

```
Subnet Mask . . . . . : 255.255.255.0
```

```
Default Gateway . . . . . : 192.168.1.1
```

4.6 Using the DebuggerTools class methods

You can use methods from the DebuggerTools system object in your User Language code to aid in the debugging process in either the *Janus Debugger* or the *Sirius Debugger*.

You can use these methods without error in User Language code running in *Model 204 Onlines* not licensed for the Debugger, as long as those Onlines are licensed at least for the *Janus SOAP* product.

The following methods are available:

- [AmDebugging](#)^[162]

- [Break](#)^[162]
- [ClientCommand](#)^[164]
- [Command](#)^[164]
- [DebugOff](#)^[165]
- [StatusMessage](#)^[166]

AmDebugging method

The AmDebugging shared method queries whether or not a User Language program is being run under the *Janus Debugger* or the *Sirius Debugger*.

The method takes no arguments, and it returns a numeric result:

```
%num = %(DebuggerTools):AmDebugging
```

where:

%num is a numeric variable that can be either of these:

- 0** The program is not running under either Debugger. This includes the case where the program runs after debugging is suspended by a [SIRIUS DEBUG SUSPEND](#)^[155] command.
- 1** The program is running under the *Janus Debugger* or under the *Sirius Debugger*.

AmDebugging lets you insert code that runs only when debugging, as in the following example:

```
b
  if ( %(DebuggerTools):AmDebugging ) then
    print 'extra debugging information'
  end if
  * normal processing .....
```

Break method

The Break shared method pauses execution on the statement that follows the Break method specification. If your User Language request is not being run under the Debugger, the Break method does not act — which lets you leave it in code. It essentially has the same effect as using the Debugger Client to put a breakpoint on the statement that would follow the Break method call.

The Break method has one, optional, argument:

```
[%statusMsg] = %(DebuggerTools):Break[(command)]
```

where:

%statusMsg is a string that contains the message in the Client [status bar](#)^[50] that results from the execution of *command*. If the request containing the method is not running in the Debugger, *%statusMsg* contains a "break is ignored" message.

command is a string expression (case not important) that contains the [Client command](#)^[179] or [macro](#)^[309] to be executed immediately following the break in execution. The command or macro in the string may be preceded by the keyword **Command** or **Macro**. This option requires at least version 7.8 of the *Sirius Mods*.

If *command* is not qualified by the keyword **Command** or **Macro**, the Debugger Client searches first for a macro named *command*, then for a Client command named *command*. You can use the **Command** or **Macro** keyword to search exclusively for a specified command or exclusively for a specific macro. The macro search is restricted to the Client installation folder or the [designated](#)^[296] macro folder.

The Break method will **not** break execution in the following cases:

- If it is invoked in a routine or method that you have elected to [step over](#)^[56]
- If you have issued a [Run Until Procedure](#)^[75], and the Break call is encountered in a procedure that precedes the "Run Until" procedure
- If a [white list](#)^[79] is enabled, and the Break method is contained in a procedure that is not on the white list

This code fragment includes a Break method invocation:

```
%status is longstring
%status = %(DebuggerTools):break('clearWatch')
* Break, execute the clearWatch command, and pause
* (awaiting instruction to execute the next statement)
%x = 666
```

When a Break method pauses execution, **DebuggerTools:break hit** is displayed in the [Status bar](#)^[50] of the Debugger Client (except, if Break executes a command, the effect of the command is displayed). Also, unlike Client-set breakpoints, the **UL>** at the beginning of the source code line does *not* change to **BR>**, and no red color highlighting is applied to the line.

You can also use the DebuggerTools ClientCommand method to execute a Client command; it does so without an execution break.

For more information about breakpoints, see [Using breakpoints](#)^[57].

ClientCommand method

The ClientCommand shared method lets you execute a [Client command](#)^[179] or [macro](#)^[309] from within a User Language request. If your request is *not* being run under the Debugger, the ClientCommand method harmlessly takes no action. The method requires version 7.8 or higher of the *Sirius Mods*.

The ClientCommand method takes one argument and returns a string value:

```
%status = %(DebuggerTools):ClientCommand([Command | Macro] command)
```

where:

%status is a string that contains the message in the Client [status bar](#)^[50] that results from the execution of the *command* parameter.

command is a string expression (case not important) that identifies the Client command or macro to execute, and it includes any parameters of the command or macro. If *command* is not qualified by the keyword **Command** or **Macro**, the Debugger Client searches first for a macro named *command*, then for a Client command named *command*. You can use the **Command** or **Macro** keyword to search exclusively for a specified command or exclusively for a specific macro. The macro search is restricted to the Client installation folder or the [designated](#)^[296] macro folder.

For example:

```
%statusMsg is longstring
%statusMsg = %(DebuggerTools):clientCommand('Addwatch %x')
%statusMsg = %(DebuggerTools):clientCommand('Macro mymacro %i')
```

After execution of the above statements, *%statusMsg* contains the string "1 watch item added".

Note: ClientCommand does **not** pause request debugging execution before (or after) it executes a Client command. If you want to produce a debugging execution pause before command execution, use the [Break](#)^[162] command with a command parameter.

Command method

The Command shared method lets you effectively execute a [SIRIUS DEBUG](#)^[151] command within a User Language request, that is, as a statement.

The Command method takes one argument and returns a numeric value:

```
%rc = %(DebuggerTools):Command(string)
```

where:

`%rc` is a numeric variable that can be either of these:

- 0** The command/method succeeded.
- <>0** The command/method failed.

string is a quoted string that specifies which SIRIUS DEBUG subcommand to execute. It may be in one of the following forms:

- The term **OFF**, **SUSPEND**, **RESUME**, or **STATUS**.
- The term **ON**, followed by the additional parameters required by the SIRIUS DEBUG ON command.
- The term **CLIENTCOMMAND**, followed by a blank, followed by its argument "clause," which consists of a [Client command](#)^[179] or [macro](#)^[309] and any parameters it requires, optionally preceded by a **Command** or **Macro** keyword:

'CLIENTCOMMAND [Command | Macro] command [cmdparms]'

If *command* is accompanied by a preceding **Command** or **Macro** keyword or by a following value for *cmdparms*, then such a clause must itself be enclosed in quotes (as shown in the second example below).

Without a qualifying **Command** or **Macro** keyword, the Debugger Client searches first for a macro named *command*, then for a Client command named *command*. The **Command** or **Macro** keyword lets you search exclusively for a specified command or exclusively for a specific macro.

Macro searches are restricted to the Client installation folder or the [designated](#)^[296] macro folder.

The CLIENTCOMMAND option requires version 7.8 or higher of the *Sirius Mods*. It is likely to be simpler to use the DebuggerTools ClientCommand method than to use the CLIENTCOMMAND option of the Command method.

In the following example, the Command method turns debugging on for the next procedure that runs; the Janus client port and Debugger client workstation are specified:

```
%rc = %(DebuggerTools):command('ON DEBsock 198.242.444.234')
```

In this example, Command sends a user interface command to the Client:

```
%rc = %(DebuggerTools):command('CLIENTCOMMAND 'AddWatch %i''')
```

DebugOff method

The DebugOff shared method turns off debugging for a thread for which debugging was turned on with a [SIRIUS DEBUG ON](#)^[151] command. It is thus a way to issue a **SIRIUS DEBUG OFF** command via a method call.

The DebugOff method takes no arguments and returns a numeric result:

```
%num = %(debuggerTools):DebugOff
```

where:

%num is a numeric variable that can be either of these:

- 0** *Sirius Debugger* debugging was not on when the DebugOff method was invoked.
- 1** *Sirius Debugger* debugging was on and is now turned off.

When debugging is on and a DebugOff method executes, debugging stops immediately, the Client/Online connection is broken, **Online has disconnected** is displayed in the [Status bar](#)^[50] of the Debugger Client, and the request completes at the Online thread.

StatusMessage method

The StatusMessage shared method contains the message in the Client [Status bar](#)^[50] that results from the last interaction with the Client. The method requires Version 7.8 or higher of the *Sirius Mods*.

The StatusMessage method returns a string value:

```
%status = %(DebuggerTools):Statusmessage
```

4.7 Using a local editor

As described in this section, you can use a local editor to view and modify most of the small text files that the Debugger Client provides for its various purposes. You can also use a local editor to interactively modify your source code procedure files.

Editing small text files

The Debugger configuration file ([debuggerConfig.xml](#)) contains an element with which you can specify a non-default text editor to work with the various small text files employed by the Debugger Client. As described in [this step](#)^[379] in the configuration of [debuggerConfig.xml](#), you use the **notepadReplacement** element to select your own local editor to replace the Client default Microsoft *Notepad* editor.

Editing procedure files

The **Procedure Line Details**^[126] dialog box in the Debugger Client provides detailed information about a source code line, such as the name of the procedure and procedure file from which it comes. You can configure the **Edit** button on the **Procedure Line Details** dialog box to copy the procedure code to an editor on your workstation.

From the editor, you can change and save the the code, and the saved file is immediately returned to the Online to replace the original procedure. To see if the change had the desired results, you re-issue from the browser the call that produced the original procedure code.

Currently, two editors are supported:

- Xtend® from Yoda Software (<http://yoda-software.com.au>)

Xtend is a GUI-based editor designed specifically to edit procedures written in *Model 204* User Language. It is aware of User Language syntax, including Sirius extensions to User Language.

Xtend transfers files to and from the *Model 204* Online using the HTTP protocol, and you must either define a *Janus Web Server* port or set up an RCL Connect* connection to use it. This document describes only the Web Server connection (which requires *Janus Web Server* authorization). For information about using RCL Connect*, you must see the *Xtend* documentation.

- UltraEdit® from IDM Computer Solutions, Inc. (<http://ultraedit.com>)

UltraEdit is a GUI-based text editor that is designed to edit program source code from a wide variety of languages.

UltraEdit transfers files to and from the *Model 204* Online using the FTP protocol, and you must define a Janus FTP Server port to use it. This means you must be authorized for *Janus Sockets*.

These sections follow:

[Using Xtend with the Debugger](#)^[168]

[Using UltraEdit with the Debugger](#)^[172]

4.7.1 Using Xtend with the Debugger

To use *Xtend* as an adjunct to your debugging, you set up a *Janus Web Server* port and an APSY subsystem in the *Model 204 Online*, as well as configure the *Xtend* GUI and the Debugger Client on your workstation. Once this configuration is complete, you can use *Xtend* with the Debugger, as described above in [Using a local editor](#)^[166].

Note: You must use version 2.11 or higher of *Xtend*. The set up details provided below are for version 2.11.

Also, as stated in in [Using a local editor](#)^[166], RCL Connect* is an alternative to *Janus Web Server*, but this document describes only the Web Server connection.

The configuration steps below are described in this section. They assume the Debugger installation has been completed and tested, as described in [Product Installation](#)^[363]. Much of the *Xtend* product installation and set up is also documented in greater detail in Help files provided with the product.

1. [Install and set up Xtend in your Model 204 Online](#)^[168]
2. [Install and set up the Xtend GUI](#)^[169]
3. [Update the Debugger Client configuration file](#)^[170]
4. [Test the configuration](#)^[172]

Install and set up Xtend in your Model 204 Online

Skip to the next subsection ([Install and set up the Xtend GUI](#)^[169]) if *Xtend* is already installed at your site.

1. From <http://www.yoda-software.com.au>, download to your workstation two Windows installation files (one for the *Xtend* GUI and one for the Model 204 subsystem and web port).
2. Set up a Model 204 subsystem (XTEND) to control *Xtend* processing.
Notes for doing this are provided in the *XtendInstall.HLP* file in the *XtendInstall* folder. See the "Xtend Apsy" section.
3. From the PC, run the *Xtend* installer executable (*XtendInstall.exe*), which will populate the XTEND subsystem files.
4. In the Online, run the XTEND.JANUS.DEFINE procedure in the XTENDPRC procedure file to start the *Xtend* Janus Web port.

The default port number for the non_SSL port is 7878; for the secure port it is 7879. Change these numbers if necessary for your site.

Install and set up the Xtend GUI

1. If you will use *Janus Debugger* and have set up a proxy server for the Internet Explorer browser, update the proxy settings to provide a bypass for connections to the *Xtend* web port.
 - a. From the **Tools** menu, select **Internet Options**; then select the **Connections** tab, and click the **LAN Settings** button.
 - b. Locate the **Proxy Server** area, and click the **Advanced** button.
 - c. In the **Exceptions** area, in the list box labeled "Do not use proxy server for addresses beginning with," specify the URL of the *Xtend* web server port you [defined](#)^[168].
 - d. Click **OK** as needed to close the multiple dialog boxes.
2. Set up a remote connection to *Model 204* from the *Xtend* GUI.
 - a. Open the *Xtend* GUI executable file (*Xtend.exe*), and select **Options** from the **View** menu.
 - b. In the **Options** dialog box, select the **Remote** tab.
 - c. In the grid in the main work area, specify a name for this connection and supply a *Model 204* user ID, host name, and the web server port you [defined](#)^[168], and click the **Apply** button.
 - d. If you will use *Sirius Debugger*, select the name (Caption) of the remote connection you just defined from the **Default edit online when using Sirius Debugger** drop-down list, then click the **OK** button.
 - e. Optionally, review and update the settings in the other **Options** tabs.
3. Add the names of the procedure files (and their privileges) whose procedures you will be editing with *Xtend*.
 - a. Select **Administrator Functions** from the **Admin** menu; then select the **Files** tab.
 - b. In the **File** columns, name the procedure files that contain the procedures you want to make accessible from *Xtend*; in the **Privileges** column, select a privilege level.

This information can alternatively be specified in the XTEND subsystem itself.
 - c. Click the **Save Details** button, followed by the **Close** button.

Note: For procedure files that have explicit passwords, you must either add the file name to the XTEND subsystem files listed in the Subsystem File Use screen in the *Model 204* Subsystem Management facility, or you must modify a subroutine in the XTENDPRC procedure file (as described in the *XtendInstall.HLP* file).

4. Test your connection.
 - a. In the **Procedure List** dialog box, select the tab that is labeled with the name of your remote connection.

A list of your procedures is retrieved and displayed.
 - b. Display a test procedure's code (double-click the procedure name), make some update, then save the updated procedure by clicking the Save File icon on the toolbar.
 - c. In your Online, check that the updated procedure has replaced the original.
 - d. If your connection test is successful, close *Xtend*.

Update the Debugger Client configuration file

Once *Xtend* is installed and configured for debugging, you enable the Debugger(s) to invoke it by adding an entry to the Debugger Client configuration file (*debuggerConfig.xml*). This file is installed in the same directory as the Debugger Client executable file, and it is [initially configured](#)^[374] as part of the Debugger Client installation.

To update the file:

1. Open the *debuggerConfig.xml* file in a text editor.
2. Add an `<editor>` element (bounded by an `<editor>` start tag and an `<\editor>` end tag) at the same level (as a sibling of) the existing `<serverList>` element.

Include these `<editor>` sub-elements, and specify values for them as described in the `Comment` section below:

Sub-element	Comment
<code><program></program></code>	<p>The identifier of the <i>Xtend</i> program executable file (case does not matter). For example: <code><program>xtend</program></code>.</p> <p>Note: If the Windows system variable <code>Path</code> does not include the folder path that points to the <i>Xtend</i> executable (for example, <code>C:\Program Files\Xtend</code>), either add it to the Windows variable now, or specify the folder path before the executable file name in the <code><program></code> value.</p> <p>To locate the <code>Path</code> variable specification on a Windows XP workstation, find the Control Panel (say, Start menu > Settings > Control Panel), then select System > Advanced tab > Environment Variables button > System variables > Path, then click Edit to see the full specification of the <code>Path</code> variable.</p>

When complete, your configuration file should have a structure like the following:

```

<debuggerConfig version="1.0">
  <serverList>
    .
    .
    .
  </serverList>
  <proxy>
    .
    .
    .
  </proxy>
  <editor>
    <program>xtend</program>
  </editor>
  .
  .
  .
</debuggerConfig>

```

3. Save and close the file.

Test the configuration

In the Debugger Client, edit a procedure.

1. Restart the Debugger Client.
2. Load a procedure in the **Source Code** tab of the Debugger Client:
 - *Janus Debugger*: From your web browser, invoke a URL that includes a procedure.
 - *Sirius Debugger*: Issue the [SIRIUS DEBUG ON](#)^[151] command; then include a procedure from the *Model 204* command line.
3. Right-click a line of code, select **Procedure Information** from the context menu, then click the **Edit** button on the **Procedure Line Details** dialog box.

Xtend should open, displaying in its working area the procedure that contains the code line that you right-clicked. If it fails to do so, and you verified earlier that, by itself, *Xtend* successfully transfers files from the *Model 204* Online, begin your troubleshooting in the Debugger Client `debuggerConfig.xml` file settings for the `<editor>` element.

4.7.2 Using UltraEdit with the Debugger

To use *UltraEdit* as an adjunct to your debugging, you must set up a Janus FTP server in the *Model 204* Online as well as configure *UltraEdit* and the Debugger Client on your workstation. Once this configuration is complete, you can use *UltraEdit* with the Debugger, as described above in [Using a local editor](#)^[166].

Note: You can use any version of *UltraEdit* that supports FTP Open. The set up details provided below are for version 12.10b and later.

The configuration steps below are described in this section. They assume the Debugger installation has been completed and tested, as described in [Product Installation](#)^[363].

1. [Set up a Janus FTP server in the Model 204 Online](#)^[172]
2. [Set up UltraEdit](#)^[173]
3. [Update the Debugger Client configuration file](#)^[175]
4. [Test the configuration](#)^[177]

Set up a Janus FTP server in the Model 204 Online

You must have the *Janus Sockets* product enabled, and you should refer to the "Janus FTP Server" chapter in the *Janus Sockets Reference Manual*.

The following steps provide a simple example of JANUS commands you can use to set up a Janus FTP Server configured to access the procedures in the MYPROCFILE file:

1. Create an FTP Server port with the JANUS DEFINE command:

```
JANUS DEFINE FTPULTRA portnum FTPSERVER 8 -  
AUDTERM -  
BINDADDR xxx.xxx.xxx.xx
```

where *portnum* will be the TCP port number for accessing your FTP Server, and *xxx.xxx.xxx.xx* is the IP address on your *Model 204* host to which the port is bound.

2. Create a mapping that provides FTP Write access to the procedure file.

The following example gives Write access to the file to all users, for the port defined in the previous step. The JANUS FTP command is described in the *Janus Sockets Reference Manual*.

In this example, MYPROCFILE is made the home folder, although it need not be (for example, if you already have a home folder set to something else).

```
JANUS FTP FTPULTRA ASSIGN /MYPROCFILE TO FILE MYPROCFILE  
JANUS FTP FTPULTRA HOME /MYPROCFILE TO ALL  
JANUS FTP FTPULTRA ALLOW /MYPROCFILE WRITE TO ALL
```

3. Start the FTP Server port:

```
JANUS START FTPULTRA
```

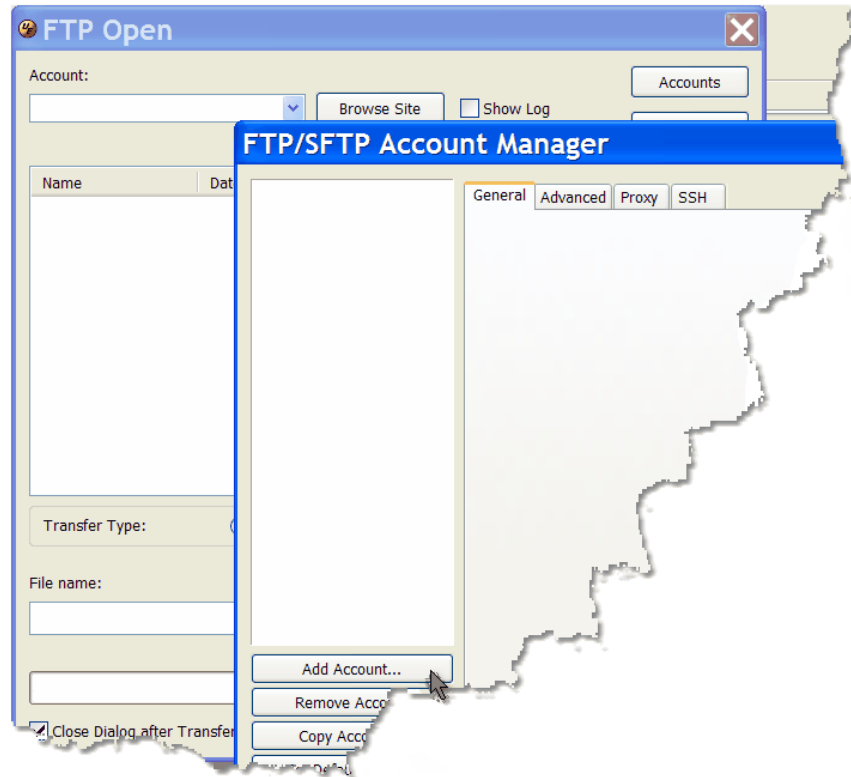
4. Issue the following command, and verify that your port is defined and started:

```
JANUS STATUS
```

Set up UltraEdit

1. Download and install a copy of *UltraEdit* on the workstation that hosts the Debugger Client.
2. Start *UltraEdit*, and in the **File** menu, select **FTP**, then **Open from FTP**.
3. From the **FTP Open** dialog box, set up an account that contains the information *UltraEdit* needs to access your procedure file via the Janus FTP Server you set up:

- a. Click the **Accounts** button, and in the **FTP/SFTP Account Manager** dialog box, click the **Add Account** button:



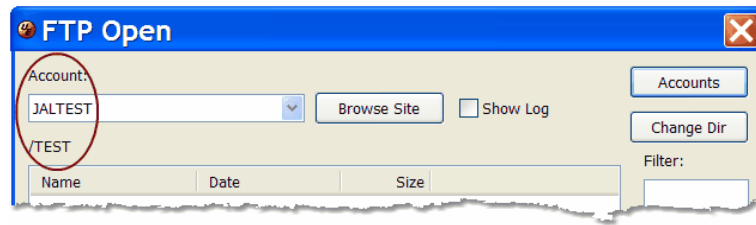
- b. Provide values for the fields that are displayed in the **General** tab:

Account	Identifies this set of FTP connection values to <i>UltraEdit</i> , and you must also specify this value in the Debugger Client configuration file.
Protocol	Leave the default value, FTP .
Server	The DNS name of the <i>Model 204</i> Online's host machine, or the IP address on your <i>Model 204</i> host to which the Janus FTP port is bound (BINDADDR in the Janus FTP Server port definition ^[172]).
Port	Replace the default value (21) with the port number you specified in the Janus FTP Server port definition ^[172] .
Username	<i>UltraEdit</i> accesses <i>Model 204</i> with this user ID and the Password value, below. Case does not matter.
Password	The password for Username , above. Case does not matter.
User Account	Any value specified here, or no value, is ignored by the Janus FTP Server.

Initial Directory Your initial location upon connecting to the FTP Server. This must correspond to a folder you specified — including the forward slash (/) separator that precedes it — in a JANUS FTP *pname* ASSIGN command in the [set up](#)^[172] of your Janus FTP Server. Case does not matter.

- c. Click the **Apply** button, then click the **OK** button.

On the **FTP Open** page, your Account name displays in the **Account** drop-down list, and your Initial Directory value displays below that list.



- d. Test your FTP connection: select the **Show Log** checkbox, then click the **Browse Site** button.

The files in your procedure file should display in the central list box.

The FTP functions represented by the various buttons on the right side of the **FTP Open** dialog box are all operational (except for **Create Dir** and **Permissions**, which Janus FTP does not support), so be cautious if you experiment further in this dialog box. For more information about what these buttons do, see the *UltraEdit* online Help.

- e. Select a default **Transfer Type** option — it will be associated with this account — then click the **Cancel** button to exit.

4. Close *UltraEdit*.

Update the Debugger Client configuration file

Once *UltraEdit* is installed and configured for debugging, you enable the Debugger(s) to invoke it by adding an entry to the Debugger Client configuration file (*debuggerConfig.xml*). This file is installed in the same directory as the Debugger Client executable file, and it is [initially configured](#)^[374] as part of the Debugger Client installation.

To update the file:

1. Open the *debuggerConfig.xml* file in a text editor.
2. Add an `<editor>` element (bounded by an `<editor>` start tag and an `<\editor>` end tag) at the same level (as a sibling of) the existing `<serverList>` element.

Include these `<editor>` sub-elements, and specify values for them as described in the `Comment` section below:

Sub-element	Comment
<code><program></program></code>	<p>The identifier of the <i>UltraEdit</i> program executable file (case does not matter). For example: <code><program>uedit32</program></code>.</p> <p>Note: If the Windows system variable <code>Path</code> does not include the folder path that points to the <i>UltraEdit</i> executable (for example, <code>C:\Program Files\Ultra-Edit-32</code>), either add it to the Windows variable now, or specify the folder path before the executable file name in the <code><program></code> value.</p> <p>To locate the <code>Path</code> variable specification on a Windows XP workstation, find the Control Panel (say, Start menu > Settings > Control Panel), then select System > Advanced tab > Environment Variables button > System variables > Path, then click Edit to see the full specification of the <code>Path</code> variable.</p>
<code><account></account></code>	<p>The name of the <i>UltraEdit</i> account you set up^[173] to connect to the Janus FTP Server folder that contains the procedures you will be debugging. For example: <code><account>JALTEST</account></code>.</p> <p>Note: This value is case-sensitive; it must exactly match the value specified in <i>UltraEdit</i>.</p>

When complete, your configuration file should have a structure like the following:

```
<debuggerConfig version="1.0">
  <serverList>
    .
    .
    .
  </serverList>
  <proxy>
    .
    .
    .
  </proxy>
  <editor>
    <program>uedit32</program>
    <account>JALTEST</account>
  </editor>
  .
  .
  .
</debuggerConfig>
```

3. Save and close the file.

Test the configuration

1. Restart the Debugger Client.
2. Load a procedure in the **Source Code** tab of the Debugger Client:
 - *Janus Debugger*: From your web browser, invoke a URL that includes a procedure.
 - *Sirius Debugger*: Issue the [SIRIUS DEBUG ON](#)^[151] command; then include a procedure from the *Model 204* command line.
3. Right-click a line of code, select **Procedure Information** from the context menu, then click the **Edit** button on the **Procedure Line Details** dialog box.

UltraEdit should open, displaying in its working area the procedure that contains the code line that you right-clicked. If it fails to do so, and you verified earlier that, by itself, *UltraEdit* successfully transfers files from the FTP Server, begin your troubleshooting in the Debugger Client `debuggerConfig.xml` file settings for the `<editor>` element.

Note: *UltraEdit* 12.10a versions open the procedure to line 1 in your procedure code. 12.10b and later versions, as well as 12.0x versions, open the procedure to the same line number as that from which you invoke the edit in the Debugger Client.

CHAPTER 5 *The Client Command Reference*

Client commands are the operations that you invoke from Client menus and can assign to a Debugger Client button, keyboard shortcut, or macro. It is intended that there be a command available for any Client operation you want to automate.

The following subsections describe individually the available commands, which are specified without regard for case.

Later sections in this document describe ways to use the commands:

- [Reconfiguring GUI buttons and hot keys](#)^[283] describes how you can map any command to a Client button or hot key.
- [Default settings of buttons and hot keys](#)^[291] lists the default hot keys and buttons with which some of these commands are associated.
- [Using Debugger Macros](#)^[309] describes how you can use script multiple commands to run consecutively.

A very few of the commands are **macro-only**: commands that may be used only in a Debugger macro. The descriptions of these commands include a **Scope** section that reminds of this restriction.

As a quick means of testing what a command does, you can open the [Command Line](#)^[317] tool and run your command from there (specifying a qualifying **Command** keyword if a same-named macro command exists). For informational, error, and trace messages from the command, you can use the [Console](#)^[316] tool.

You can also execute a command from within a User Language request by using the **ClientCommand** method of the [DebuggerTools](#)^[161] class.

5.1 **addWatch command**

Action [Adds to the Watch Window](#)^[88] the item currently specified in the [Entity-name input box](#)^[51].

Syntax:

addWatch [*item*]

where *item* is the name of the item to be added (one of [these](#)^[87]). In a macro, this argument is required.

Client menu Data Display > Add Watch

Introduced Build 26

5.2 **addWatchOnCurrentLine command**

Action [Adds to the Watch Window](#)^[88] any variables found in the current **Source Code** line.

Syntax:

addWatchOnCurrentLine

Client menu Data Display > Add Watch on Current Line

Introduced Build 28

5.3 **assert command**

Action Performs an equality or inequality comparison between a) the value of program data or a [macro variable](#)^[319] or [Client function](#)^[321], and b) a constant or the value of a macro variable or macro function. For example:

assert %i=666

This command lets you create simple testing macros that ensure that key elements in your code have the values you expect. If the comparison expression you construct with `assert` is not logically true, you receive a failure message. If true, you receive no confirmation.

`assert` failure messages are displayed in the [console](#)^[316], if it is open. Otherwise, they are displayed in a Windows message box. They have the following format:

Assert failed: failing_assert_statement

For example:

Assert failed: assert &b = "no way"

The `assert` command syntax follows:

```
ASSERT &var | %xxx | g.xxx | f.xxx | $listcnt(x)
      | $listinf(x,y) | &&function
      = | <>
      string | [-]nnn | &var | &&function
```

where:

- `&var` is a previously defined macro variable.
- `%xxx` is a mainframe variable.
- `g.xxx` is a Debugger [global variable reference](#)^[96].
- `f.xxx` is a Debugger [field reference](#)^[95], possibly with a subscript.
- `$listcnt/$listinf` are the Debugger functions for [viewing \\$list counts and elements](#).^[98]
- `&&function` is a macro function.
- `<>` is an inequality operator (as of Client Build 59).
- `string` is a quoted string constant (double-quotes or single quotes are valid).
- `[-]nnn` is an integer constant with an optional leading minus sign.

Here are examples of valid `assert` statements:

```
assert g.JACK = "No play makes Jack a dull boy."
```

```
ASSert %s='Hey Moe'
```

```
ASSERT $listcnt(%g)=2
```

```
ASSERT $listinf(%g,2) = " makes Jack a dull boy."
```

`assert &this = 'that'`

`assert &this <> &that`

Notes:

- The [&&assertFailureCount](#)^[324], [&&assertSuccessCount](#)^[325], and [&&assertStatus](#)^[324] Client functions, as well as [&&globalAssertFailureCount](#)^[332], [&&globalAssertSuccessCount](#)^[333], and [&&globalAssertStatus](#)^[333], report `assert` command results.
- If you are specifying a Client [mapping command](#)^[288] and your assertion includes an ampersand character (&) or the not-equal operator (<>), you must XML entity-encode the character. For example, to map the command `assert &i<>6`, you specify it like this:

```
<mapping command="assert &amp;i&lt;&gt;6" button="button9"/>
```

For Client builds before 54, `assert` is allowed only in Debugger [macros](#)^[309] and is not available as a mappable Client command.

Client menu —

Introduced Build 28

5.4 bottom command

Action

Scrolls to the bottom of the current tab; has no effect if the [Proc Selection](#)^[13] page is being displayed.

Optionally, scrolls to the bottom of *the Client window you specify* as the value of the command's **In window** prefix.

Syntax:

[In window] bottom

where:

window is one of the following keywords, not case sensitive, which identify a Client window. Links are provided to help identify the less common windows:

[about](#)^[39]
[auditTrail](#)
[commands](#)^[38]
[console](#)
[executionHistory](#)^[134]
[executionTrace](#)
[keyboardShortcuts](#)^[38]
[source](#)
[textviewer](#)^[149]
[value](#)^[101]
[watchWindow](#)
[webBuffer](#)

Client menu **Window > Bottom**

Introduced —

5.5 **breakOnNextProc** command

Action Interrupts [White List](#)^[82] or [Run Until](#)^[78] processing. The next procedure will be debugged.

Syntax:

breakOnNextProc

Client menu —

Introduced Build 29

5.6 **breaks** command

Action Sets breakpoints on lines after comments that have the form ***Break** (see [Setting multiple breakpoints at once](#)^[60]).

Syntax:

breaks

Note: The search for the ***Break** lines begins from the current line, so you probably should precede the **breaks** command by a [top](#)^[274] command if used in a macro.

Client menu **Breakpoints > Breaks**

Introduced —

5.7 **breaksAt** command

Action Sets breakpoints on lines that match a search string (see [Setting multiple breakpoints at once](#)^[60]).

Syntax:

breaksAt [*string*]

where *string* is the search string or [regular expression](#).^[45] The default is to use the value in the [Search text box](#).^[39]

Note: The search for the lines on which to set breakpoints begins from the current line, so you probably should precede **breaksAt** by a [top](#)^[274] command if used in a macro.

In a macro (only), you must explicitly specify the string at which **breaksAt** is to break.

Client menu Breakpoints > Breaks At

Introduced —

5.8 **buttonBar** command

Action Opens and determines the position of the Client's main [button bar](#)^[39] in a Client [external window](#).^[300] To locate the button bar within the Client main window, use the [mainButtonBar](#)^[223] command.

Syntax:

buttonbar [*position*]

where *position* is one of these options that control the disposition of the external button bar (the **Button Bar** window):

- **top** docks the window (places it, immobile) at the top left corner of the desktop.
- **bottom** docks the window at the bottom left corner of the desktop.
- **float** places the window centrally and non-docked (mobile) on the desktop.
- **hide** closes the window.
- **show** opens the window and restores the previous dock mode, if any.
 - If the main button bar is not currently an external window, **show** makes it external and restores its previous dock mode, if any, or uses **float** as the default mode.
 - If the main button bar is already in an external window, **show** has no effect.

The **show** parameter, new in Build 56, is the default.

Client menu **Window > Show Button Bar in External Window**

Introduced Build 55

5.9 **cancel command**

Action [Cancels](#)^[65] the User Language request that is being, or about to be, debugged. Gives a "do you really want to" prompt.

Syntax:

cancel

Client menu Execution > Cancel

Introduced —

5.10 **clearAudit command**

Action Clears the Audit Trail display.

Syntax:

clearAudit

Client menu Window > Clear Audit Trail

Introduced —

5.11 **clearBreakpointOnCurrentLine** command

Action [Clears a breakpoint](#)^[60] on the currently selected line in the **Source Code** (or [Daemon](#)^[141]) page; if the currently selected line is not an executable statement, clears a breakpoint on the next executable line after the currently selected line.

Syntax:

clearBreakpointOnCurrentLine

Notes:

- If you have not explicitly selected a code line, the current line is the highlighted code line in the current execution position.
- If you execute this command for a line that already has no breakpoint set, no additional action is taken.

For code lines for which a breakpoint is set, the [toggleBreakpointOnCurrentLine](#)^[272] command has the same effect as the **clearBreakpointOnCurrentLine** command.

To set a breakpoint on the current line, you can use the [setBreakpointOnCurrentLine](#)^[257] command or the **toggleBreakpointOnCurrentLine** command.

To clear all breakpoints, you can use the [clearBreaks](#)^[184] command.

Client menu —

Introduced Build 57

5.12 `clearBreaks` command

Action	Removes all previously set breakpoints ^[57] . To remove an individual breakpoint, you can use the clearBreakpointOnCurrentLine ^[188] command. Syntax: clearBreaks
Client menu	Breakpoints > Clear All Breakpoints
Introduced	—

5.13 `clearButton` command

Action	Removes all previously set mappings ^[285] for a particular Client button. Syntax: clearbutton <i>buttonName</i> where <i>buttonName</i> is one of: <ul style="list-style-type: none"> button0, button1, ... button14, the names of the main button bar^[39] buttons extraButton0, extraButton1, ... extraButton14, the names of the extra button bar^[43] buttons <p>After you execute the command, the button is removed immediately from the button bar to which it applies but not from the button-mapping file (ui.xml or uimore.xml) if it was specified there. A restart of the Client restores any mappings you remove with clearButton, if they were in the ui.xml or uimore.xml file.</p> <p>See also the mapButton^[224] command.</p>
Client menu	—
Introduced	Build 57

5.14 **clearExecutionTrace** command

Action Clears the [Execution Trace](#)^[13] display.

Syntax:

clearExecutionTrace

Client menu **Window > Clear Execution Trace**

Introduced Build 37

5.15 **clearHistory** command

Action Clears the [Execution History window](#).^[134]

Syntax:

clearHistory

Client menu —

Introduced Build 50

5.16 **clearKey** command

Action Removes all previously set [mappings](#)^[285] for a particular Client keyboard-key combination.

Syntax:

clearKey *key*

where *key* is a keyboard key: a single letter or digit, or one of **F2** through **F12** (the function keys)

Immediately after you execute the command, the keyboard shortcut loses effect but its mapping is **not** removed from the button-mapping file ([ui.xml](#) or [uimore.xml](#)) if it was specified there. A restart of the Client restores any mappings you remove with **clearKey**.

See also the [mapKey](#)^[226] and [clearbutton](#)^[189] commands.

Client menu —

Introduced Build 57

5.17 **clearMacroConsole** command

Action Clears the [Console window](#).^[316]

Syntax:

clearMacroConsole

Client menu —

Introduced Build 50

5.18 **clearStatus** command

Action Clears current message (error or informational), if any, from the Client [Status bar](#)^[50].

Syntax:

clearStatus

This command be useful to reduce confusion when developing macros: at the start of the macro, you can clear any earlier messages, so you will know that any subsequent messages are from the macro under development.

Client menu —

Introduced Build 58

5.19 **clearWatch** command

Action [Removes](#)^[90] all watched items from the Watch Window.

Syntax:

clearWatch

Client menu Data Display > Clear Watch

Introduced —

5.20 **clearWebBuffer** command

Action Clears the [Web Buffer page](#)^[12].

Syntax:

clearWebBuffer

Client menu Window > Clear Web Buffer

Introduced Build 43

5.21 **closeCommandLine** command

Action Closes the [Command Line dialog box](#)^[317] (entitled **Macro Command Line** prior to Client build 53), which lets you run a macro by entering its name and any parameters.

Syntax:

closeCommandLine

See also [openCommandLine](#)^[231].

Client menu —

Introduced Build 53

5.22 **closeExternalAuditTrailWindow** command

Action Closes an [external Audit Trail window](#)^[300].

Syntax:

closeExternalAuditTrailWindow

Client menu —

Introduced Build 50

5.23 **closeExternalButtonWindow** command

Action Closes an [external Button Bar window](#).^[42]

Syntax:

closeExternalButtonWindow

Client menu —

Introduced Build 54

5.24 **closeExternalExecutionTraceWindow** command

Action Closes an [external Execution Trace window](#).^[300]

Syntax:

closeExternalExecutionTraceWindow

Client menu —

Introduced Build 50

5.25 **closeExternalWatchWindow** command

Action Closes an [external Watch Window](#).^[300]

Syntax:

closeExternalWatchWindow

Client menu —

Introduced Build 50

5.26 **closeExternalWebBufferWindow** command

Action Closes an [external Web Buffer window](#).^[300]

Syntax:

closeExternalWebBufferWindow

Client menu —

Introduced Build 50

5.27 **closeExternalWindows** command

Action Closes any open Client [external Windows](#).^[300]

Syntax:

closeExternalWindows

Client menu **Window > Close External Windows**

Introduced Build 50

5.28 **closeHistory** command

Action Closes the [Execution History window](#).^[134]

Syntax:

closeHistory

Client menu —

Introduced Build 50

5.29 **closeMacroConsole** command

Action	<p>Closes the macro console^[316].</p> <p>Syntax:</p> <p>closeMacroConsole</p> <p>See also openMacroConsole.^[234]</p>
Client menu	—
Introduced	Build 53

5.30 **closeValueDisplay** command

Action	<p>Closes the current Value window^[101], if any; takes no action if no Value window is open.</p> <p>Syntax:</p> <p>closeValueDisplay</p> <p>A Value window is used by multiple Client operations and commands to display or expand values (for example, the expandList^[205], expandObject^[206], pafgi^[234], pai^[235], and valueDisplay^[279] commands).</p>
Client menu	—
Introduced	Build 50

5.31 `continueIf` command

Action Determines whether the processing of a Debugger [macro](#)^[309] may continue. If the state ("True" or "False") of a specified command argument ([macro variable](#)^[319], [client function](#)^[321], or constant) is **True**, the macro continues. As of Build 62, `continueIf` can evaluate the truth of an expression (equality or inequality) involving macro variables, client functions, or constants.

If the state of the `continueIf` argument or comparison expression is **False**, the macro containing this command **and any macro(s) within which this macro is contained** terminate (without error). This behavior contrasts with that of the [continueMacroIf](#)^[198] command, which exits only the macro containing the `continueMacroIf` command.

Syntax:

```
continueIf test
```

and prior to Build 62 *test* is:

```
&var | &&function | const
```

and as of Build 62 *test* is:

```
&var | &&function | const [ = | <> &var | &&function | const ]
```

Where:

- `&var` is a macro variable that may or may not already exist.
- `&&function` is a Client function. Requires Build 58 or higher.
- `const` is a constant. Requires Build 58 or higher.
- `=` and `<>` are equality and inequality operators, respectively.

Notes:

If `continueIf` is used with a single argument and that argument is `0` or a zero length (null) string, or if it is undefined, its state is considered to be **False**. For all other values, its state is considered to be **True**.

In an equality comparison, a null string compared to a null string is **True**, and any undefined item makes an equality comparison **False**. Each of these truth outcomes is reversed in an inequality comparison.

Example:

```
# Execute Step and optionally update history
nospan
step
continueIf &historyWanted
getHistory
...
continueIf &testMe <> &testMe2
```

Scope Allowed only in Debugger macros; **not** available as a mappable Client command

Client menu —

Introduced Build 37

5.32 **continueMacroIf** command

Action Determines whether the processing of the Debugger [macro](#)^[309] that contains this command may continue. If the state ("True" or "False") of a specified command argument ([macro variable](#)^[319], [client function](#)^[321], or constant) is **True**, the macro continues. As of Build 62, **continueMacroIf** can evaluate the truth of an expression (equality or inequality) involving macro variables, client functions, or constants.

If the state of the command argument is **False**, the macro terminates (without error). This behavior contrasts with that of the [continueIf](#)^[197] command, which exits not only the macro containing the **continueIf** command but also any macro(s) within which that macro is contained.

Syntax:

```
continueMacroIf test
```

and prior to Build 62 *test* is:

```
&var | &&function | const
```

and as of Build 62 *test* is:

```
&var | &&function | const [ = | <> &var | &&function | const ]
```

Where:

- **&var** is a macro variable that may or may not already exist.
- **&&function** is a Client function. Requires Build 58 or higher.

- `const` is a constant. Requires Build 58 or higher.
- `=` and `<>` are equality and inequality operators, respectively.

Notes:

If `continueMacroIf` is used with a single argument and that argument is `0` or a zero length (null) string, or if it is undefined, its state is considered to be `False`. For all other values, its state is considered to be `True`.

In an equality comparison, a null string compared to a null string is `True`, and any undefined item makes an equality comparison `False`. Each of these truth outcomes is reversed in an inequality comparison.

Example:

```
# See if it is watched, done if not
in watchWindow searchFromTop &argstring
continueMacroIf &&searchSuccess
#
# It was watched, remove and tell what we did
removeCurrentWatch
echo &&concatenate("removed watch ", &argstring)
...
continueMacroIf &testMe = 1
...
continueMacroIf &&blackOrWhiteList <> 'black'
```

Scope Allowed only in Debugger macros; **not** available as a mappable Client command

Client menu —

Introduced Build 57

5.33 **copy command**

Action	Copies to the clipboard the lines currently visible in the active tabbed page ^[10] .
	Syntax: copy
Client menu	Window > Copy
Introduced	—

5.34 **createMacro command**

Action	Lets you name (via a Windows dialog box) and edit (via Windows <i>Notepad</i>) a new blank Debugger macro ^[309] file.
	Syntax: createMacro
Client menu	Macros > New Macro
Introduced	Build 26

5.35 debugPreview command

Action When the [Source Preview feature](#)^[85] is enabled, triggers a full download of the program source code for normal debugging. If the program has compilation errors, the full compilation error listing is downloaded.

Syntax:

debugPreview

Client menu Execution > Debug Previewed Source

Introduced Build 30

5.36 decrement command

Action Decreases the value by 1 of a specified numeric [macro variable](#)^[319]. If the macro variable is non-numeric or is undefined, an error is issued.

Syntax:

decrement *&var*

where *&var* is a macro variable that may or may not already exist.

The inverse of this command is [increment](#)^[215]. A related Client function is [&&sum](#)^[341].

Client menu —

Introduced Build 57

5.37 **disableButton** command

Action Disables the (currently enabled) [button bar button](#)^[39] that you specify. A disabled button performs no action and has gray text. You might want to disable a button to simplify the button bar display for a particular context.

Syntax:

disableButton *buttonName*

where *buttonName* is *button0*, *button1*, etc; that is, the name of a currently [mapped](#)^[285] button.

If the command executes successfully, you receive a **Disabled button: *buttonName*** message, and the button's label changes from black text to gray.

See also the [enableButton](#)^[204] command.

Introduced Build 62

5.38 echo command

Action Displays a message to the user.

Syntax:

```
echo message
```

where *message* is one of these:

- A non-quoted string that contains the message.
- A Client [function](#)^[321]. The result of the function execution is displayed as the message.

The message string starts with the first non-blank character after the **echo** keyword, and it continues as far as the end of the line. For example:

```
echo Have a nice day!
```

The message is normally displayed in a standard Windows informational box (entitled **Macro message**). If the [macro console](#)^[316] is open, however, the message is sent to the console instead.

Note: If you are specifying a Client [mapping command](#)^[288] and your message includes an ampersand character (&), you must XML entity-encode the character. For example, to map the command **echo &foo**, you specify it like this:

```
<mapping command="echo &amp;foo" button="button13"/>
```

Client menu —

Introduced Build 27

5.39 **editMacroFromUISelection** command

Action Lets you select and open for editing (via *Windows* file-selection dialog) an existing Debugger [macro](#)^[309] file.

Syntax:

editMacroFromUISelection

Client menu **Macros > Edit Macro**

Introduced Build 53

5.40 **enableButton** command

Action Enables the (currently disabled) [button bar button](#)^[39] that you specify. A disabled button performs no action and has gray text.

The **enableButton** command affects only buttons that are disabled because they were specified in a previous [disbleButton](#)^[202] command. If a button is disabled because its action is not appropriate in the current debugging context, **enableButton** does **not** enable the button. For example, after you cancel a request, the **Cancel** and **Clear Breaks** buttons are disabled, and an **enableButton** command for one of these buttons has no effect: the button remains disabled and no return message is displayed.

Syntax:

enableButton *buttonName*

where *buttonName* is **button0**, **button1**, etc; that is, the name of a currently [mapped](#)^[285] button.

If the command executes successfully, you receive an **Enabled button: *buttonName*** message, and the button's label changes from gray text to black.

Introduced Build 62

5.41 **evaluate** command

Action Constructs and runs a Client command. The **evaluate** argument values are concatenated into a single string and run as a Client command.

Syntax:

```
evaluate {&var/'string1'/"string2"} ...
```

where:

- *&var* is a previously defined [macro variable](#)^[319].
- *string1* and *string2* are single- or double-quoted string literals.

No blanks are placed between the argument values when they are concatenated, so you may have to provide for them within quoted string values. If a macro variable is not defined, or if the string that is built is not a valid command, an error is issued.

Here is a macro definition that makes heavy use of the **evaluate** command:

```
# histButtons.macro: Assign extrabutton<extrBtn>-<extrBtn+3>
# for history traversal
# Usage: macro historyButtons <extrBtn>
extraButtonbar main
set &bnum = &argstring
evaluate 'mapButton extrabutton' &bnum ' previousHistory'
increment &bnum
evaluate 'mapButton extrabutton' &bnum ' nextHistory'
increment &bnum
evaluate 'mapButton extrabutton' &bnum ' firstHistory'
increment &bnum
evaluate 'mapButton extrabutton' &bnum ' lastHistory'
evaluate 'echo buttons ' &argstring '-' &bnum ' set for history'
```

Client menu —

Introduced Build 57

5.42 **expandList** command

Action [Displays](#)^[104] in a **Value** window the list items in the \$list, Stringlist, or Arraylist referenced by the variable specified as the command argument.

Same as the **List Display** context menu option for **Watch Window** items, including how to control the number of items displayed.

For example:

expandList %ls

If the command argument does not reference a \$list, Stringlist, or Arraylist, you receive an error message.

If issued by a macro, *and* the [Macro Console](#)^[316] is open, then the value is displayed in the **Macro Console** window.

Client menu —

Introduced Build 43

5.43 **expandObject** command

Action [Displays](#)^[111] in a **Value** window a list of the class Variable names and values of its required object instance argument. Same as the **Expand Object** context menu option for **Watch Window** items.

For example:

expandObject %scout

If the command argument does not reference an object, or if the object does not have class variables, you receive an error message.

If issued by a macro, *and* the [Macro Console](#)^[316] is open, the value is displayed in the **Macro Console** window.

Client menu —

Introduced Build 43

5.44 **extraButtonBar** command

Action Opens and determines the position of a Client's [extra button bar](#)^[43]. Command options locate the bar in an [external window](#)^[300] or on the Client main window, merged with the main button bar.

Syntax:

extraButtonbar [*position*]

where *position* is one of these options that control the disposition of the extra button bar (the **Extra Buttons** window):

- **top** docks the window (places it, immobile) at the top left corner of the desktop.
- **bottom** docks the window at the bottom left corner of the desktop.
- **float** places the window centrally and non-docked (mobile) on the desktop.
- **main** adds the extra buttons to the main button bar, immediately following the last main button. Available in Client build 57. This is equivalent to selecting the **Extra Buttons** option in the [Preferences](#)^[18] dialog box.
- **hide** closes the the **Extra Buttons** window.
- **show** opens the window and restores the previous dock mode, if any, or uses **float** as the default mode.

If the **Extra Buttons** window is already open, **show** has no effect.

The **show** parameter is the default.

Client menu **Window > Show Extra Button Bar Window**

Introduced Build 56

5.45 **feoDisplay command**

Action	<p>Displays FOR EACH OCCURRENCE OF (FEO) statement information^[116] (current OCC subscript value) for the current source line (if it is an FEO statement). If the current source line is not an FEO statement, an error is issued.</p> <p>This command is equivalent to right-clicking a source line and selecting FEO OCC IN Value from the context menu.</p> <p>Syntax:</p> <p style="text-align: center;">feoDisplay</p>
Client menu	—
Introduced	Build 58

5.46 **firstHistory command**

Action	<p>Scans chronologically backward in the current statement execution history^[134], then highlights <i>in the Source Code or Daemon tab</i> the first (earliest) statement in the history.</p> <p>Syntax:</p> <p style="text-align: center;">firstHistory</p>
Client menu	Execution > Select First History Line
Introduced	Build 54

5.47 focusToSearchBox command

Action Gives the input focus to the [Search text area](#)^[44]. Once that area has focus, pressing the Enter key invokes the [searchDown](#)^[247] command, so you can repeat a search using only the keyboard.

Syntax:

focusToSearchBox

By default, the Ctrl+F key combination also gives focus to the Search text area.

Client menu —

Introduced Build 29

5.48 generatePac command

Action Generates a PAC (Proxy Auto Config) JavaScript file from the Debugger configuration settings in `debuggerConfig.xml`. If this file is defined to the Internet Explorer browser, IE will use the Debugger Client as a proxy *only* for requests for the hosts (Onlines) [specified](#)^[374] in `debuggerConfig.xml`.

The IE browser does this host filtering if the PAC file location is specified for it in the **Address** value in **Tools > Internet options > Connections > LAN settings > Use automatic configuration script**.

By default, `generatePac` merges the code it generates with that of an existing PAC file (if such a file is already specified in the Internet Explorer configuration options).

Syntax:

generatePac [*file* [*overwrite|nomerge*]]

Where *file* is the name of a file which, if not specified, defaults to `debuggerPac.js`. The generated file is placed by default in the Client [work-file folder](#)^[296]. If no such work folder is configured, the Client installation folder is used.

If you specify a *file* value:

- You can use quotation marks to indicate an absolute or relative Windows file-system path:
 - If quoted (for example, `generatePac "c:\pac\debuggerPac.js"`), the file value is treated as an absolute file path.
 - If not quoted (for example, `generatePac foo.js`), the value is treated as a path relative to the work-file folder.
- You can also use a file URL (for example, `file://c:\xxx\yyy.js`) to specify the file.

If a file with the same name as *file* already exists, it will not be overwritten, unless you specify `overwrite`. If you use `overwrite`, the *file* value must be explicitly specified.

Note: If your *file* specification contains an error (typo, incorrect file or path name, etc.), the Internet Explorer browser ignores the command and does not inform you of the error.

When `generatePac` runs, it reports its activity in the Debugger Client [console](#)^[316]. For example:

```
Command: generatePac "c:\pac\debuggerPac.js" overwrite
Generating Pac file: c:\pac\debuggerPac.js
Adding: "*198.242.244.47:3224*"
Adding: "*sirius-software.com*"
Adding: "*sirius-software.com:3224*"
Adding: "*sirius-software.com:3666*"
Adding: "*sirius-software.com:3667*"
Adding: "*sirius-software.com:3000*"
Adding: "*sirius-software.com:3669*"
Adding: "*sirius-software.com:3668*"
Adding: "*sirius-software.com:3777*"

```

The hosts in the `Adding` lines are copied from the `debuggerConfig.xml` file.

See also the [setIEmode](#)^[258] command, which combines the `generatePac` functionality with automatic specification and removal of the PAC file in the IE configuration settings when the Client starts and closes.

Client menu —

Introduced Build 62

5.49 **getHistory** command

Action [Displays a history](#)^[134] (in an **Execution History** window or in the **Execution Trace** page) of the statements executed thus far during program evaluation. The history includes calls and returns for methods and subroutines (as many as 1000 statements).

Syntax:

getHistory

See also the [previousHistory](#)^[237], [nextHistory](#)^[228], [firstHistory](#)^[208], and [lastHistory](#)^[220] commands, which let you view highlighted history statements within the program in the **Source Code** tab (or **Daemon** tab).

Client menu **Execution > Get/Display History**

Introduced Build 34

5.50 **getVariablesForClass** command

Action [Displays](#)^[111] a list of the names (not values) of the variable members in the system or user class that is specified as the command argument.

Similar to [expandObject](#)^[206], but does not scan the code for the values of a particular object instance.

For example:

getVariablesForClass cats

If the class in the command argument is not found, or if the class does not have member variables, you receive an error message.

If issued by a macro, *and* the [Macro Console](#)^[316] is open, then the value is displayed in the **Macro Console** window.

Client menu —

Introduced Build 43

5.51 help command

Action

Displays the Debugger's Windows Help file, or as of Builds 59 and 60, displays a brief description of a specified Client [command](#)^[179] or [function](#)^[321], or displays a list of the available user preference options or of the file types that the Debugger ignores (that is, does not display for debugging).

Syntax:

```
help [command | function | preference | ignoredFiles]
```

Where these are the parameter options:

- Specify a Client command or function (without regard for case).
- Specify **preference** to display the user preferences available to you as options of the [setPreference](#)^[260] command. This option is available as of Build 60.
- Specify **ignoredFiles** to display a list of the [filtered file types](#)^[376]. This option is available as of Build 60.

If you issue **help preference**, output like the following is displayed:

Targets for setPreference:

```
breakAfterReadScreen  
caseSensitiveAssert  
debuggerDirectives  
historyToTrace  
ieAuto  
ignoredFileTypeList  
ignoreMacroErrors  
initExclude  
macroAutorun  
pauseAtEndEval  
stopOnAssertFailure  
useProcLists  
useRoutineLists  
valueDisplayOnConsole
```

If you issue **help ignoredFiles**, output like the following is displayed:

```
Ignored file types (if ignoredFileTypeList is on):  
css,gif,htc,ico,jpeg,jpg,js,png,xml,xsl
```

Client menu **Help > Help Topics**

Introduced —

5.52 hideLower command

Action [Hides the lower section](#)^[305] of the Client main window. This is useful where the **Audit Trail** and **Watch Window** are in separate windows on the same or another monitor.

This command is equivalent to selecting the **Hide Lower Section** option of the **Main Window Options** section of the [Preferences](#)^[18] dialog box.

Syntax:

hideLower

hideLower has no effect if the lower windows are hidden when the command is issued.

See also:

- The [toggleLower](#)^[273] command also hides the lower windows if they are *not* hidden when the command is issued, but which restores the windows if they were hidden.
- The [restoreLower](#)^[243] command restores the lower windows if they were hidden.
- The [openExternalWatchWindow](#)^[233] command lets you access the **Watch Window** if the lower section of the main window is hidden.

Client menu **File > Preferences > (Main Window Options) Hide Lower Section**

Introduced Build 57

5.53 `include` command

Action	<p>Calls another Debugger macro^[309].</p> <p>Syntax:</p> <pre><code>include macroname</code></pre> <p>where <i>macroname</i> is the non-quoted name of the macro.</p>
Scope	Allowed only in Debugger macros ^[309] ; not available as a mappable Client command.
Client menu	—
Introduced	—

5.54 `ifdef` command

Action	<p>Conditionally calls another Debugger macro^[309]. The specified macro is included if and only if the state ("True" or "False") of a specified macro variable^[319] is True. If the state of the macro variable is False, no action is taken and the macro continues.</p> <p>If the variable is <code>0</code> or a zero length (null) string, or if it is undefined, its state is considered to be False. For all other values, its state is considered to be True.</p> <p>Syntax:</p> <pre><code>ifdef &var macroName</code></pre> <p>where:</p> <ul style="list-style-type: none">• <code>&var</code> is a macro variable that may or may not already exist.• <i>macroName</i> is the non-quoted name of a macro.
Scope	Allowed only in Debugger macros; not available as a mappable Client command.
Client menu	—
Introduced	Build 57

5.55 **increment** command

Action Increases the value by 1 of a specified numeric [macro variable](#)^[319]. If the macro variable is non-numeric or is undefined, an error is issued.

Syntax:

increment *&var*

where *&var* is a macro variable that may or may not already exist.

For an example that uses the **increment** command, see the [evaluate](#)^[205] command.

The inverse of this command is [decrement](#)^[201]. A related Client function is [&&sum](#)^[341].

Client menu —

Introduced Build 57

5.56 **jumpToLine** command

Action

Transfers control to a specified request statement in the **Source Code** or **Daemon** page, then executes that statement.

The command's required argument is a number or keyword that indicates the target statement:

jumpToLine [**current** | *number*]

where:

- The keyword **current** specifies a jump to and then execution of the current statement.
- *number* may be in one of three forms:

nnn | *-nnn* | *+nnn*

- *nnn* specifies an "absolute jump" to the *nnn* [statement line number](#)^[11] in the **Source Code** or **Daemon** page display, then an execution of that statement.
- *-nnn* or *+nnn* specifies a "relative jump," jumping the indicated number of statements backward or forward relative to the current (yellow highlighted) line, followed by the execution of that statement.

For example, specifying **-1** re-executes the statement prior to the current line. **+1** skips the current executable statement and executes the one following it.

If you [use the command in a macro](#)^[309]:

1. When you specify the **jumpToLine** command in the macro, explicitly supply its argument (the **current** keyword or a number to indicate the target line) or specify an [argument variable](#).^[314]
2. Observe the [jump validation rules](#).^[84]

If you [use the command in a mapped button or hot key](#)^[283]:

1. Do not specify an argument for the **jumpToLine** command in the mapping; you specify the argument (the **current** keyword or a number to indicate the target line) in the [Entity-name input box](#)^[51].
2. Observe the jump validation rules.

For information about invoking a jump by right-clicking a line in the **Source Code** or **Daemon** tab, see [Altering the flow of execution](#).^[83]

Client menu

—

5.57 **jumpToMatch** command

Action

Transfers control to a request statement that contains a specified matching string, then attempts to execute that statement.

Syntax:

jumpToMatch *string*

where *string* is the string for which a match is searched *from the top* (first) line in the **Source Code** or **Daemon** page. The search string may be a regular expression, as described for the [Search button](#)^[39].

The nesting level of a statement has no effect on whether it is located.

If a match is not found, **JumpToMatch string not found** is displayed in the [Status area](#)^[50]. If a match is found but the statement is not executable, **Invalid line for jump** is displayed in the **Status** area.

If you [use the command in a macro](#)^[309]:

1. When you specify the **JumpToMatch** command in the macro, explicitly supply its argument (the **current** keyword or a number to indicate the target line) or specify an [argument variable](#)^[314].
2. Observe the [jump validation rules](#)^[84].

If you [use the command in a mapped button or hot key](#)^[283]:

1. Do not specify an argument for the **JumpToMatch** command in the mapping; you specify the argument (the **current** keyword or a number to indicate the target line) in the **Search** text box.
2. Observe the jump validation rules.

For information about invoking a jump by right-clicking a line in the **Source Code** or **Daemon** tab, see [Altering the flow of execution](#)^[83].

Client menu

—

Introduced

Build 27

5.58 **kill** command

Action Stops a running [macro](#)^[309] and issues a message indicating that fact. If **kill** is issued and the specified macro is not running, you receive a message indicating that no such macro is running. This is most suitable if a macro's execution spans multiple requests.

Syntax:

kill *macroName*

where *macroName* is the name of the macro you want to stop.

Client menu **Macros**

Introduced Build 58

5.59 **labelButton command**

Action Lets you replace the label of a Client [button bar button](#)^[39].

Syntax:

```
labelButton buttonName newlabel
```

where:

- *buttonName* is *button0*, *button1*, etc; that is, the name of a currently [mapped](#)^[285] button.
- *newlabel* can be as many as 50 characters (the button is expandable).

If the command executes successfully, the label changes immediately, and you receive a **Label set for: *buttonName*** message.

Example:

The following macro uses **labelButton**:

```
continueMacroIf &&blackOrWhiteList = 'black'  
labelButton button0 BlackList On  
turnOffBlackList  
restoreTitle  
set &changed = 1  
clearStatus
```

Introduced Build 62

5.60 **lastHistory** command

Action Scans chronologically forward in the current statement [execution history](#)^[134], then highlights *in the Source Code or Daemon tab* the last (latest) statement in the history.

Syntax:

lastHistory

Client menu Execution > Select Last History Line

Introduced Build 54

5.61 **loadWatch** command

Action Restores from a local **.watch** file a list of items to display in the **Watch Window**, as described in [Saving and restoring Watch Window contents](#).^[91]

Syntax:

loadWatch [*watchfile*]

where the command's optional argument is the name of the **.watch** file to be loaded. If you omit *watchfile*, a Windows file selection dialog box lets you select a saved **.watch** file when the command executes.

Otherwise, if you specify a *watchfile* argument (the **.watch** extension may be omitted), the named file opens when the command executes.

To locate the watch file you identify, the Client looks in the folder that contains the **JanusDebugger.exe** file, [by default](#)^[296]. If the watch file cannot be found, an error is issued.

You may also specify the watch file name in these ways:

- As a [macro variable](#)^[319]. For example:

```
set &a = "foo"  
loadWatch &a
```
- As part of an absolute file system path, which *must be enclosed in quotation marks*. Requires Build 58 or higher. For example:

```
loadWatch "c:\temp\foo.watch"
```

The [saveWatch](#)^[246] command saves the **Watch Window** contents to a watch file.

Client menu **Data Display > Load Watch**

Introduced Build 49

5.62 **macro command**

Action Identifies a [user-defined macro](#)^[309].

Syntax:

macro *macroName*

where *macroName* is the name of the macro you are identifying.

Client menu —

Introduced Build 26

5.63 **macroConsole command**

Action Invokes the [macro console](#)^[316], which reports the starting and completing of macro execution, as well as any error messages.

Syntax:

macroConsole

A synonym for **macroConsole** is [openMacroConsole.](#)^[234]

See also [closeMacroConsole.](#)^[196]

Client menu **Macros > Console**

Introduced Build 43

5.64 **macroTrace** command

Action Starts or stops a display in the [macro console](#)^[316] of a trace of all macro statements in your debugging session.

Syntax:

```
macroTrace [ on | off ]
```

where:

- **on** indicates that macro lines will be traced until turned off with **macroTrace off**.
- **off** turns off **macroTrace**, if it is on.

Note: When **macroTrace** is turned on, the trace output is produced *only if* the macro console is open.

This is an example of macro trace output:

```
>>>macroTrace: step
>>>macroTrace: assert %i = 1
>>>passed...
>>>macroTrace: run
>>>macroTrace: step
>>>macroTrace: assert %i = 2
>>>passed...
```

Note that the trace includes the pass or fail status of [assert](#)^[180] commands, and it includes syntax help if commands have syntax errors (as of Client Build 59).

Client menu —

Introduced Build 50

5.65 macroWait command

Action	Slows down Debugger macro execution by adding a wait after each command in the macro ^[309] . Syntax: <code>macroWait <i>nnn</i></code> where: <ul style="list-style-type: none"><i>nnn</i> is the number of milliseconds to wait after the execution of each command in the macro.
Scope	Allowed only in Debugger macros; not available as a mappable Client command
Client menu	—
Introduced	Build 57

5.66 mainButtonBar command

Action	Determines the position within the Client window (non-external) of the Client's main button bar ^[39] . To place the button bar in an external window, use the buttonBar ^[186] command. Syntax: <code>mainButtonBar <i>position</i></code> where <i>position</i> is one of these options: <ul style="list-style-type: none">top places the bar in its default location, above the Client main window.center places the bar below the main window (but above the search, tracing, and value displaying controls).bottom places the bar at the very bottom of the Client window. The show parameter, new in Build 56, is the default.
Client menu	File > Preferences > Main Button Bar
Introduced	Build 57

5.67 manual command

Action Displays the PDF reference manual for the Debugger.

Syntax:

manual

Client menu Help > View PDF Manual

Introduced —

5.68 mapButton command

Action Lets you [assign a command to a Client button](#)^[283] without having to edit a mapping file (`ui.xml` or `uimore.xml`) and to restart the Client.

Syntax:

mapButton [buttonModifier-]buttonName command

Where (case not important):

- *buttonModifier* is one of:
 - **Alt**
Maps the Alt-key version of the button (command runs when you click the button while holding down Alt)
 - **Cntrl, Control, Cntl, Ctl, or Ctrl**
Maps the control-key version of the button (command runs when you click the button while holding down Ctrl)

buttonModifier is optional, and its default is no modifier (command runs when you click the button).
- *buttonName* is one of:
 - **button0, button1, ... button14**, the names of the [main button bar](#)^[39] buttons
 - **extraButton0, extraButton1, ... extraButton14**, the names of the [extra button bar](#)^[43] buttons

- *command* is either:
 - A [Client command](#)^[179]
 - The **separator** keyword (case not important), which converts the *buttonName* button to a visual [separator button](#)^[285]

Examples:

```
mapbutton button0 viewtext
```

```
mapButton ctl-button0 showAbout
```

```
mapButton ALT-button0 showCommands
```

```
mapButton eXTRAbutton0 openmacroConsole
```

After you execute the command, the new mapping is reflected immediately in the button bar to which it applies. The new mapping does **not** appear in the button-mapping file. The new mapping does **not** survive a Client restart.

The command to remove a button mapping is [clearButton.](#)^[189] The command to map a keyboard shortcut is [mapKey.](#)^[226]

Client menu —

Introduced Build 56

5.69 mapKey command

Action Lets you [assign a keyboard shortcut](#)^[284] without having to edit a mapping file (`ui.xml` or `uimore.xml`) and to restart the Client.

Syntax:

mapKey [modifier-]key command

Where (case not important):

- *modifier* is one of:
 - **Alt**
Maps the Alt-key version of a key combination (command runs when you hold down the Alt key and press the specified key)
 - **Cntrl**, **Control**, **Cntl**, **Ctl**, or **Ctrl**
Maps the control-key version of a key combination*modifier* is optional, and its default is no modifier.
- *key* is a keyboard key: a single letter or digit, or one of **F2** through **F12** (the function keys)
- *command* is a [Client command](#)^[179]

Examples:

mapkey f2 Step

mapKey alt-f2 stepover

mapkey Ctl-f2 stepout

Mapkey ALT-2 run

After you execute the command, the new mapping is reflected immediately in the keyboard. The new mapping does **not** appear in the mapping file, and it does not remain if the Client is restarted.

The command to remove a keyboard shortcut mapping is [clearKey.](#)^[191] The command to map a Client button is [mapButton.](#)^[224]

Client menu —

Introduced Build 57

5.70 **moveBrowserToTop** command

Action Brings the web browser window to the top of your current stack of open windows.

Syntax:

moveBrowserToTop

Formerly available for cases where the Client was paused while awaiting user input (for example, READ SCREEN or \$WEB_FORM_DONE processing), this window stack manipulation is invoked at any time by executing **moveBrowserToTop**. Like the window feature for READ SCREEN or \$WEB_FORM_DONE code, you set up **moveBrowserToTop** processing by [specifying your browser's program title](#)^[144] in the **Preferences** dialog box in the Debugger Client.

Client menu —

Introduced Build 54

5.71 **moveTn3270ToTop** command

Action Brings the window of your 3270 terminal-emulator program to the top of your current stack of open windows.

Syntax:

moveTn3270ToTop

Formerly available for cases where the Client was paused while awaiting user input (for example, READ SCREEN processing), this window stack manipulation is invoked at any time by executing **moveTn3270ToTop**. Like the window feature for READ SCREEN code, you set up **moveTn3270ToTop** processing by [specifying your emulator's program title](#)^[63] in the **Preferences** dialog box in the Debugger Client.

Client menu —

Introduced Build 54

5.72 **nextCompileError** command

Action	<p>Finds the next line in error relative to the current line, if the request being displayed fails to compile^[138].</p> <p>Syntax:</p> <p>nextCompileError</p> <p>You can map this command to a button that also has another command or macro mapped to it. See Button toggle for compilation errors^[290].</p>
Client menu	Error > Next Compile Error
Introduced	—

5.73 **nextHistory** command

Action	<p>Scans chronologically forward in the statement execution history^[134], then highlights <i>in the Source Code or Daemon tab</i> the statement that was executed immediately following the statement that is currently highlighted with the Execution Position color.^[294]</p> <p>Syntax:</p> <p>nextHistory</p>
Client menu	Execution > Select Next History Line
Introduced	Build 54

5.74 noSpan command

Action	<p>Reverses the default spanning behavior of a Debugger macro^[309]. By default, macro execution spans the evaluation of requests (if a request completes before a macro is finished, the remaining macro commands apply to the next request).</p> <p>With noSpan in effect, any running macro is terminated at the end of request evaluation.</p> <p>Syntax:</p> <p style="text-align: center;">noSpan</p> <p>The span^[269] command reverses a previously issued noSpan command.</p>
Scope	Allowed only in Debugger macros ^[309] ; not available as a mappable Client command
Client menu	—
Introduced	Build 37

5.75 nsLookup command

Action Requests a host name or IP address from the Domain Name System (DNS). This is useful for debugging setup issues such as getting the IP number of the Client workstation host or for testing the Client's ability to resolve a host name. It is similar to the operating system nsLookup command.

Syntax:

nslookup [host]

Where *host* is the name of a TCP/IP host machine known to your local DNS server. Specifying a *host* value displays that host's IP number. If *host* is not specified, the IP number of the workstation on which the Client is running is displayed.

Examples:

- Specifying:

nslookup google.com

produces the following output:

**Looking up IP number for: google.com
IP Number is: 74.125.226.233**

- Specifying:

nsLookup

produces the following output:

**Looking up this workstation's IP number
IP Number is: 198.242.244.3**

- Specifying an unknown host results in an error message:

**Looking up IP number for: googlasdasd
DNS lookup failure for: googlasdasd: No such host is known**

Client —
menu

IntroduBuild 61
ced

5.76 openCommandLine command

Action Invokes the **Command Line** dialog box, which lets you run a macro or command. Any macro you identify must be located in the same folder as the Debugger Client executable file or in a folder [specified](#)^[296] in the `debuggerConfig.xml` file.

Syntax:

openCommandLine

See also [closeCommandLine](#)^[193].

Client menu **Macros > Command Line**

Introduced Build 53

5.77 openExternalAuditTrailWindow command

Action Displays in a separate, [external window](#)^[300] the current contents of the Client **Audit Trail** tab. Or, it brings to the top of your current stack of open windows the external **Audit Trail** window.

Syntax:

openExternalAuditTrailWindow

Client menu **Window > Open External Audit Trail Window**

Introduced Build 50

5.78 **openExternalButtonWindow** command

Action Displays in a separate, [external window](#)^[300] the current contents of the Client's main [button bar](#)^[39] (restoring any previous docking position). Or, if the main button bar is already open, it brings to the top of your current stack of open windows the external **Button Bar** window.

Syntax:

openExternalButtonWindow

A [buttonBar show](#)^[186] command has very similar effects.

Client menu **Window > Show Main Button Bar in External Window**

Introduced Build 54

5.79 **openExternalExecutionTraceWindow** command

Action Displays in a separate, [external window](#)^[300] the current contents of the Client [Execution Trace tab](#)^[13]. Or, it brings to the top of your current stack of open windows the external **Execution Trace** window.

Syntax:

openExternalExecutionTraceWindow

Client menu **Window > Open External Execution Trace Window**

Introduced Build 50

5.80 **openExternalWatchWindow** command

Action Displays in a separate, [external window](#)^[300] the current contents of the Client Watch Window. Or, it brings to the top of your current stack of open windows the external Watch Window.

Syntax:

openExternalWatchWindow

Client menu Window > Open External Watch Window
Data Display > Open External Watch Window

Introduced Build 49

5.81 **openExternalWebBufferWindow** command

Action Displays in a separate, [external window](#)^[300] the current contents of the Client [Web Buffer tab](#)^[12]. Or, it brings to the top of your current stack of open windows the external Web Buffer window.

Syntax:

openExternalWebBufferWindow

Client menu Window > Open External Web Buffer Window

Introduced Build 50

5.82 **openMacroConsole** command

Action Invokes the [console](#)^[316], which reports the starting and completing of macro or command execution, as well as any error messages.

Syntax:

openMacroConsole

The **openMacroConsole** command is a synonym for the [macroConsole](#)^[221] command.

See also [closeMacroConsole.](#)^[196]

Client menu **Macros > Console**

Introduced Build 53

5.83 **pafgi** command

Action Displays all the visible fields for the current *Model 204* field group, just like the User Language PAFGI statement.

Syntax:

pafgi

Note: The debugging context must be a field group (for example, within an FEO Of Fieldgroup loop), the version of the *Sirius Mods* must be at least 7.6, and the version of *Model 204* must be at least 7.2.

For more information, see [Displaying a record's field groups.](#)^[120]

Client menu **Data Display > PAFGI**

Introduced Build 46

5.84 **pai command**

Action Displays all the visible fields for the current *Model 204* record just like the User Language PAI statement.

Syntax:

pai

Note: The debugging context must be a record, and the version of the *Sirius Mods* must be at least 7.6.

For more information, see [Displaying all fields in a record.](#)¹¹⁷

Client menu Data Display > PAI

Introduced Build 45

5.85 pin command

Action Pins the specified Client [external window\(s\)](#)^[300], that is, keeps the window(s) at the top of the Client PC's open window stack. Such a window can be moved by mouse around the screen, and it can be joined by other pinned windows. It remains at the top as long as it is open, even if other external windows or applications are subsequently opened. It can only be removed from the top by closing it, by unpinning it (the [unpin](#)^[278] command), or by clicking its upper-right-corner "Minimize" button.

Syntax:

```
pin {windowname | pattern | *}
```

where you must specify one of these:

- *windowname*, the (case not important) name, or title, at the top of a Client work window or external window
- *pattern*, a character sequence that ends with an asterisk (*), which performs a "wildcard" search (for example, *ab** finds the **About** window)
- A lone asterisk (*), which pins all open external windows

If you *pin* a window that is already pinned, the command is ignored. If the command indicates a window that is missing or invalid, or if it does not match an open external window, an error message is issued.

Client menu Context menu option (**Pin**) of external window title bar

Introduced Build 56

5.86 preferences command

Action Displays the Preferences dialog box.

Syntax:

```
preferences
```

Client menu **File > Preferences**

Introduced —

5.87 `previousCompileError` command

Action Finds the previous line in error relative to the current line, if the request being displayed [fails to compile](#)^[138]. If the program you are debugging has no compilation errors, the commands does nothing.

Syntax:

`previousCompileError`

This command is unusual in that you can map it to a button that also has another command or macro mapped to it. See [Button toggle for compilation errors](#)^[290].

Client menu Error > Previous Compile Error

Introduced —

5.88 `previousHistory` command

Action Scans chronologically backward in the statement [execution history](#)^[134], then highlights *in the Source Code or Daemon tab* the statement that was executed immediately prior to the statement that is currently [highlighted with the Execution Position color](#).^[294]

Syntax:

`previousHistory`

Client menu Execution > Select Previous History Line

Introduced Build 54

5.89 reloadBlackList command

Action Updates the existing [Black List](#)^[79] with the current contents of the `blacklist.txt` file, so you can dynamically update your Black List. Same as clicking the **Reload Black List** button on the **Proc Selection** page.

Syntax:

reloadBlackList

Client menu Execution > Reload Black List

Introduced Build 62

5.90 reloadWhiteList command

Action Updates the existing [White List](#)^[79] with the current contents of the `whitelist.txt` file, so you can dynamically update your White List. Same as clicking the **Reload White List** button on the **Proc Selection** page.

Syntax:

reloadWhiteList

Client menu Execution > Reload White List

Introduced Build 28

5.91 reloadLists command

Action Loads or reloads to the Client [the Exclude/Include lists](#)^[70], that is, lists of procedures or routines (methods, or User Language "complex" subroutines) whose code you want to be [excluded from interactive debugging](#)^[67]. These lists must be specified in particular text files: `excludeProc.txt`, `excludeRoutine.txt`, `includeProc.txt`, and `includeRoutine.txt`.

Syntax:

reloadLists

Same as clicking the **Reload Proc/Routine/Method Lists** button on the **Proc Selection** page.

Client menu —

Introduced Build 58

5.92 **removeCurrentWatch** command

Action

Removes the currently selected [Watch Window](#)^[88] item. If there is no currently selected item, it does nothing.

The current selected item may simply be the last added or clicked line (and not highlighted), or it may be a highlighted line as the result of a search or location (top or bottom) command.

Using **removeCurrentWatch** to remove a watched item is essentially the same operation as right-clicking an item and selecting **Remove** from the context menu.

Syntax:

removeWatch

This "removeWatch" macro is an example of using the command:

```
# RemoveWatch.macro
#
# Usage: removeWatch varName
# where varName is name of watched variable to remove
# as it appears in the Watch Window.
# For example, removeWatch %j
#
# See if it is watched. If not, done:
in watchWindow searchFromTop &argstring
continueMacroIf &&searchSuccess
#
# It was watched, so remove and report:
removeCurrentWatch
echo &&concatenate("removed watch ", &argstring)
```

Client menu —

Introduced Build 58

5.93 `resetAssertCounts` command

Action Normally, the assert counts accessible from the `&&assertFailureCount`, `&&assertStatus`, and `&&assertSuccessCount` [Client functions](#)^[321] are reset when the Client is started/restarted and when a new macro is invoked. If you want to clear them at any arbitrary time, executing `resetAssertCounts` resets them to 0.

Syntax:

`resetAssertCounts`

Client menu —

Introduced Build 56

5.94 `resetGlobalAssertCounts` command

Action Normally, the assert counts accessible from the `&&globalAssertFailureCount`, `&&globalAssertStatus`, and `&&globalAssertSuccessCount` [Client functions](#)^[321] are reset only when the Client is started/restarted. If you want to clear them at any arbitrary time, executing `resetGlobalAssertCounts` resets them to 0.

Syntax:

`resetGlobalAssertCounts`

Client menu —

Introduced Build 57

5.95 **restart command**

Action	Restarts the Debugger Client. <ol style="list-style-type: none">1. Terminates all socket connections between the Client and the Online, and/or between the Client and a web browser.2. Shuts down the Client normally, as if the Exit option were selected, and closes the Client window.3. Starts the Client, reopening its window and invoking the same processing as if started by a click of the Client desktop icon (which includes reading the <code>debuggerConfig.xml</code> file). Syntax: restart
Client menu	File > Restart
Introduced	Build 56

5.96 **restartDefault command**

Action	Restarts the Debugger Client, restoring the window dimensions with which the Client displayed when it was initially installed. Otherwise, the Client restarts with the size and position (including internal window dimensions) it occupied upon last exit. Syntax: restartDefault
Client menu	File > Restart with Default Window Size
Introduced	Build 56

5.97 **restoreLower** command

Action Restores the display of the [lower section](#)^[14] of the Client main window, after it was hidden by a previous [hideLower](#)^[213] or [toggleLower](#)^[273] command or by selecting the **Hide Lower Section** option of the **Main Window Options** section of the [Preferences](#)^[18] dialog box.

Syntax:

restoreLower

This command is equivalent to clearing the **Hide Lower Section** option of the **Preferences** dialog box.

restoreLower has no effect if the lower section is *not* hidden when the command is issued.

See also the [toggleLower](#)^[273] command, which also restores the lower section if it is hidden when the command is issued, but which hides the section if it was not hidden.

Client menu **File > Preferences > (Main Window Options) Hide Lower Section**

Introduced Build 57

5.98 **restoreTitle** command

Action Restores to the default the title of a Client main window that was changed by the [setTitle](#)^[264] command. The default main window title is "The Janus Debugger" or "The Sirius Debugger."

Syntax:

restoreTitle

See also [&¤tTitle](#)^[330] and [&&originalTitle](#)^[337].

Client menu —

Introduced Build 62

5.99 **run command**

Action Executes the code displayed in the Client's **Source Code** page until end-of-request or until an error, breakpoint, or daemon is encountered.

Syntax:

run

Client menu **Execution > Run**

Introduced —

5.100 **runMacroFromUISelection command**

Action Lets you select and open for editing (via *Windows Explorer*) an existing Debugger [macro](#)^[309] file.

Syntax:

runMacroFromUISelection

Client menu **Macros > Run Macro**

Introduced Build 53

5.101 **runUntil command**

Action Runs program code without interruption until it reaches a specific procedure, then displays that procedure for debugging. Same as the [Run Until Procedure button](#)^[75].

Syntax:

rununtil *targetProc*

where *targetproc* is the name of, or a character pattern for, the target procedure.

Client menu Execution > Run Until Proc

Introduced Build 26 (macros only); Build 28 (menu)

5.102 **runUntilVariableChanges command**

Action Steps through the program being debugged, stopping if a statement modifies the value of the variable specified in the text box above the **Watch Window**. Displays in the **Execution Trace** tab the statement that modified the variable and the new variable value. Same as the [Run to Change button](#)^[133].

In a macro, the command syntax is:

rununtilVariablechanges *variable*

where *variable* is the name of the User Language variable being observed.

Client menu Execution > Run Until Variable Changes

Introduced Build 26

5.103 **runWithoutDaemons** command

Action Runs until end-of-request, or until an error or breakpoint is encountered; [does not pause at daemon code](#)^[142].

Syntax:

runWithoutDaemons

Client menu Execution > Run Without Daemons

Introduced —

5.104 **saveWatch** command

Action Saves to a local file the list of items currently displayed in the **Watch Window**, as described in [Saving and restoring Watch Window contents](#)^[91].

The command's first optional argument is the name of the **.watch** file in which to store the **Watch Window** contents:

saveWatch [*watchfile* [**overwrite**]]

If you omit *watchfile*, a Windows file-selection dialog box lets you create a **.watch** file when the command executes.

Otherwise, if you specify a *watchfile* argument (the **.watch** extension may be omitted), the **Watch Window** contents are stored in the named file when the command executes.

The Client saves the watch file in the folder that contains the **JanusDebugger.exe** file, [by default](#).^[296]

If a watch file with the same name already exists when **saveWatch** executes, the result depends on whether you also specified the **overwrite** parameter:

- If it is not specified, the **saveWatch** command fails, and an error message is issued.
- If it is specified, the existing file gets overwritten.

You may also specify the watch file name in these ways:

- As a [macro variable](#)^[319]. For example:

```
set &a = "foo"
saveWatch &a overwrite
```

- With an absolute file-system path, which *must be enclosed in quotation marks*. Requires Build 58 or higher. For example:

```
saveWatch "c:\temp\foo.watch"

saveWatch "c:\temp\foo.watch" overwrite
```

The [loadWatch](#)^[220] command restores the **Watch Window** contents from a watch file.

Client menu Data Display > Save Watch

Introduced Build 49

5.105 searchDown command

Action [Searches down](#)^[46] (relative to the current line) in the current main window page for the search string you specify in the command. Repeating the command locates the next occurrence of the search string. The search is *not* case-sensitive.

Optionally, searches down for the search string *in the Client window you specify* as the value of the command's **In window** prefix.

Syntax:

```
[In window] searchDown search_string
```

where:

- *window* is one of the following keywords, not case sensitive, which identify a Client window. Links are provided to help identify the less common windows:

[about](#)^[39]
[auditTrail](#)
[commands](#)^[38]
[console](#)
[executionHistory](#)^[134]
[executionTrace](#)
[keyboardShortcuts](#)^[38]
[source](#)
[textviewer](#)^[149]
[value](#)^[101]
[watchWindow](#)
[webBuffer](#)

- `search_string` is the required search string, with no additional quotation marks.

To locate a target string that begins with an ampersand (&) and is **not** the name of a macro variable or Client function, prefix the target with a backslash (\) to treat the ampersand as a literal.

To locate a single backslash character, escape the target backslash with a second backslash (`searchDown \\`).

The backslash escape is valid as of Client Build 58.

Command examples:

`in WatchWindow searchDown %y`

`searchDown &a`
(searches for value of macro variable `&a`)

`in auditTrail searchdown \&&myProc`
(searches for string '`&&myProc`')

Related commands include: [searchUp](#)^[251], [searchFromTop](#)^[250], [searchFromBottom](#)^[248]

Client menu Search > Search Down

Introduced —

5.106 `searchFromBottom` command

Action [Searches up](#)^[45], from the bottom of the current main window page, for the search string you specify in the command. Repeating the command locates the same occurrence of the search string. The search is *not* case-sensitive.

Optionally, searches from the bottom of the *Client window* you specify as the value of the command's **In** *window* prefix.

Syntax:

```
[In window] searchFromBottom search_string
```

where:

- window* is one of the following keywords, not case sensitive, which identify a Client window. Links are provided to help identify the less common windows:

[about](#)^[39]
auditTrail
[commands](#)^[38]
console
[executionHistory](#)^[134]
executionTrace
[keyboardShortcuts](#)^[38]
source
[textviewer](#)^[149]
[value](#)^[101]
watchWindow
webBuffer

- search_string* is the required search string, with no additional quotation marks.
- To locate a target string that begins with an ampersand (&) and is **not** the name of a macro variable or Client function, prefix the target with a backslash (\) to treat the ampersand as a literal.
 To locate a single backslash character, escape the target backslash with a second backslash (**searchFromBottom** \\\).

The backslash escape is valid as of Client Build 58.

Command examples:

```
in WatchWindow searchFromBottom printText
```

```
searchFrombottom &a  
(searches for macro variable &a)
```

```
in auditTrail searchFromBottom \&&myProc  
(searches for string '&&myProc')
```

Related commands include: [searchUp](#)^[251], [searchDown](#)^[247], [searchFromTop](#)^[250]

Client menu Search > Search From Bottom

Introduced Build 11

5.107 **searchFromTop** command

Action [Searches](#)^[45] down, from the top of the current main window page, for the search string you specify in the command. Repeating the command only locates the same occurrence of the search string. The search is *not* case-sensitive.

Optionally, searches from the top of *the Client window you specify* as the value of the command's **In window** prefix.

Syntax:

[In window] searchFromTop search_string

where:

- *window* is one of the following keywords, not case sensitive, which identify a Client window. Links are provided to help identify the less common windows:

[about](#)^[39]
auditTrail
[commands](#)^[38]
console
[executionHistory](#)^[134]
executionTrace
[keyboardShortcuts](#)^[38]
source
[textviewer](#)^[149]
[value](#)^[101]
watchWindow
webBuffer

- *search_string* is the required search string, with no additional quotation marks.
- To locate a target string that begins with an ampersand (&) and is **not** the name of a macro variable or Client function, prefix the target with a backslash (\) to treat the ampersand as a literal.

To locate a single backslash character, escape the target backslash with a second backslash (**searchFromTop **).

The backslash escape is valid as of Client Build 58.

Command examples:

in WatchWindow searchFromTop printText

searchFromTop &a

(searches for value of macro variable &a)

in auditTrail searchFromtop \&&myProc

(searches for string &&myProc)

Related commands include: [searchUp](#)^[251], [searchDown](#)^[247],
[searchFromBottom](#)^[248]

Client menu Search > Search From Top

Introduced —

5.108 searchUp command

Action [Searches up](#)^[46] (relative to the current line) in the current main window page for the search string you specify in the command. Repeating the command locates the next occurrence of the search string. The search is *not* case-sensitive.

Optionally, searches up for the search string *in the Client window you specify* as the value of the command's **In window** prefix.

Syntax:

[In window] searchUp search_string

where:

- *window* is one of the following keywords, not case sensitive, which identify a Client window. Links are provided to help identify the less common windows:

[about](#)^[39]

auditTrail

[commands](#)^[38]

console

[executionHistory](#)^[134]

executionTrace

[keyboardShortcuts](#)^[38]

source

[textviewer](#)^[149]

[value](#)^[101]

watchWindow

webBuffer

- `search_string` is the required search string, with no additional quotation marks.
- To locate a target string that begins with an ampersand (&) and is **not** the name of a macro variable or Client function, prefix the target with a backslash (\) to treat the ampersand as a literal.

To locate a single backslash character, escape the target backslash with a second backslash (`searchUp \\`).

The backslash escape is valid as of Client Build 58.

Command examples:

```
in WatchWindow searchUp printText
```

```
searchUp &a  
(searches for value of macro variable &a)
```

```
in auditTrail searchup \&&myProc  
(searches for string '&&myProc')
```

Related commands include: [searchDown](#)^[247], [searchFromTop](#)^[250], [searchFromBottom](#)^[248]

Client menu Search > Search Up

Introduced Build 11

5.109 selectAuditTab command

Action Displays the [Audit Trail tab](#)^[10] in the Client main window.

For example:

```
<mapping command="selectAuditTab" key="1"  
keyModifier="ctrl" />
```

Client menu —

Introduced Build 46

5.110 **selectExecutionTraceTab** command

Action Displays the [Execution Trace tab](#)^[13] in the Client main window.

Syntax:

selectExecutionTraceTab

Client menu —

Introduced Build 46

5.111 **selectNextTab** command

Action Displays in the Client main window the page whose [tab](#)^[10] is to the right of the currently displayed page; wraps from the rightmost page to the leftmost.

Syntax:

selectNextTab

Client menu —

Introduced Build 46

5.112 **selectProcSelectionTab** command

Action Displays the [Proc Selection tab](#)^[13] in the Client main window.

Syntax:

selectProcSelectionTab

Client menu —

Introduced Build 46

5.113 **selectSourceTab** command

Action Displays in the Client main window the page ([Source Code](#)^[11] or [Daemon](#)^[141]) that contains the source code that is currently executing.

Syntax:

selectSourceTab

Client menu —

Introduced Build 46

5.114 **selectWatchWindow** command

Action Displays and gives focus either to an [external](#)^[300] **Watch Window** or to the Client main window (if no external **Watch Window** is deployed).

Syntax:

selectWatchWindow

Client menu —

Introduced Build 51

5.115 **selectWebBufferTab** command

Action Displays the [Web Buffer page](#)^[12] (if *Janus Debugger*) in the Client main window.

Syntax:

selectWebBufferTab

Client menu —

Introduced Build 46

5.116 set command

Action

Lets you create and initialize [macro variables](#)^[319]. Variable names begin with a single ampersand (&), and eligible variable values include constants, User Language variables, fields, and \$list elements, and other macro variables and [functions](#)^[321].

Syntax:

```
set &target = string | [-]nnn | &var | %xxx | g.xxx | f.xxx
              | $listcnt(x) | $listinf(x,y)
              | &&function([args])
```

where:

- *&target* is the macro variable being set.
- *string* is a quoted string constant (double-quotes or single quotes are valid).
- *[-]nnn* is an integer constant with an optional leading minus sign.
- *&var* is a previously defined macro variable.
- *%xxx* is a mainframe variable.
- *g.xxx* is a Debugger [global variable reference](#)^[96].
- *f.xxx* is a Debugger [field reference](#)^[95], possibly with a subscript.
- *\$listcnt/\$listinf* are the Debugger functions for [viewing \\$list counts and elements](#)^[98].
- *&&function* is a macro function.

Note: To set a macro variable or macro function in a Client [mapping command](#)^[288], you need to XML entity-encode each ampersand (&). For example, to map the command `set &foo = %i`, you specify it like this:

```
<mapping command="set &amp;foo = %i" button="button14"/>
```

Client menu —

Introduced Build 28

5.117 **setBlackList** command

Action

Lets you dynamically specify or clear a procedure [Black List](#)^[79], which identifies the outer procedures you do *not* want the Debugger to debug. These procedures get executed but their code is not sent to or displayed in the Client. This command temporarily overrides any existing Black List, but it does **not** physically affect the contents of the black list file stored on disk (`blacklist.txt`). The command's effect is not persistent over runs of the Client.

Syntax:

setBlackList [*proc*] ...

where *proc* is one of possibly multiple, blank-separated, procedure file names. Name matching is case insensitive, and [wildcards](#)^[76] are allowed.

If no names are specified, the current black list, if any, is treated as empty. If one or more procedure names are specified, the contents of the black list are overridden by this new set of procedure names.

You might use this command for testing:

1. Put it in a User Language procedure.
2. Send it to the Client via the [SIRIUS DEBUG CLIENTCOMMAND](#)^[151] *Model 204* command.

Doing this lets you have a mainframe-based testing harness that sets up a list on the Client. For example:

```
SIRIUS DEBUG CLIENTCOMMAND 'setBlackList P.SHEMP'  
SIRIUS DEBUG CLIENTCOMMAND 'turnOnBlackList'
```

See also the [setWhiteList](#)^[265] command.

Client menu —

Introduced Build 62

5.118 **setBreakpointOnCurrentLine** command

Action

Sets a [breakpoint](#)^[58] on the currently selected line in the **Source Code** (or **Daemon**^[141]) page; if the currently selected line is not an executable statement, sets a breakpoint on the next executable line after the currently selected line.

Syntax:

setBreakpointOnCurrentLine

Notes:

- If you have not explicitly selected a code line, the current line is the highlighted code line in the current execution position.
- If you execute this command for a line that already has a breakpoint set, no additional action is taken.

For code lines for which no breakpoint is already set, the [toggleBreakpointOn](#)^[272] command has the same effect as the **setBreakpointOnCurrentLine** command, and both commands have the same effect as double-clicking a code line or right-clicking a line and selecting **Toggle Breakpoint** from the context menu.

Example:

This macro looks for a line that matches its argument; if it finds such a line, it sets a breakpoint on it:

```
selectsourcetab
searchFromTop &argstring
continueMacroIf &&searchSuccess
setBreakPointOnCurrentLine
set &message = &&concatenate("breakpoint on line matching '", {
setStatusMessage &message
```

To clear a breakpoint that is set on the current line, you can use the [clearBreakpointOnCurrentLine](#)^[188] command or the **toggleBreakpointOn** command.

See also the [breaksAt](#)^[185] and [breaks](#)^[184] commands.

Client menu

—

Introduced

Build 57

5.119 **setIEmode** command

Action

Controls whether, and the host URLs for which, the Debugger will serve as the [proxy server](#)^[380] for users of the Internet Explorer browser. The command has options to set or remove the setting of the Debugger Client as the proxy server for IE browser requests. The command also has an option to create a [PAC \(Proxy Auto Config\) file](#)^[385], which restricts the host URLs for which the Client will act as proxy server, and an option to merge such a PAC file with a preexisting PAC file, if any.

Syntax:

setIEmode {none|proxy|newPac|mergedPac}

Where:

- **none**
Makes no changes to Internet Explorer's configuration-tool settings, and undoes any IE setting changes made since the Debugger Client was started.

This option also undoes the effects of any other **setIEmode** commands issued since the Client was started.
- **proxy**
Makes the Debugger Client the proxy server for Internet Explorer. This is the same as issuing the command **setPreference ieAuto on** or setting the **Preferences** dialog box equivalent, the **proxy** option for **IE Mode**.
- **newPac**
Generates a PAC file using the Debugger configuration file (**debuggerConfig.xml**) settings, and sets IE to use it. Any existing PAC file (that is, specified in IE in the **Address** value in **Tools > Internet options > Connections > LAN settings > Use automatic configuration script**) is *not* used.
- **mergedPac**
Merges any existing PAC file with a PAC file generated from the Debugger configuration file settings, and sets IE to use it.

For parameter options that modify the IE **Tools** configuration, no restart of the IE browser is necessary, and the Debugger provides automatic maintenance (makes the IE **Tools** modifications at Client startup, then removes them when the Client closes).

Example:

If you issue the following command, the Debugger Client immediately generates a merged PAC file and configures IE to run it; then at Client shutdown, the command restores IE's initial (pre-command) settings:

setIEmode mergedPac

Once the above command is issued, IE runs the script each time a URL is requested, and the Debugger Client serves as proxy only for the hosts specified in the script. The merged PAC file specification in IE will persist through Client closes and starts until you use **setIEmode** to change or remove it.

See also:

- The [generatePac](#)^[209] command, which creates and/or merges a PAC file, but makes no modifications to the IE configuration settings.
- The [showIE](#)^[267] command, which displays the current IE operating mode plus additional details.

Client menu —

Introduced Build 62

5.120 **setM204Data** command

Action

Lets you set a *Model 204* %variable or global variable as if you right-clicked a **Watch Window** item and selected [Change Value](#)^[124]. The behavior and restrictions for this command are the same as the **Change Value** option.

Syntax:

```
setM204Data %xxx | g.xxx  
= "string" | [-]nnn | &var | &&function([args])
```

where:

- %xxx is a User Language variable.
- g.xxx is a Debugger [global variable reference](#)^[96].
- *string* is a quoted string constant (double-quotes or single quotes are valid).
- [-]nnn is an integer constant with an optional leading minus sign.
- &var is a previously defined macro variable.
- &&function is a macro function.

Note: This command may only be issued while a request is being evaluated.

Client menu —

Introduced Build 50

5.121 **setPreference** command

Action

Lets you use a [macro](#)^[309] or [mapped](#)^[285] button or key to control Client preference settings that are available via the [Preferences](#)^[18] dialog box or the [Proc Selection](#)^[13] page.

Syntax:

```
setPreference option {on|off}
```

where *option* is one of the following:

- **breakAfterReadScreen**
Selects or clears the [Break after READ SCREEN](#)^[61] checkbox in the Client's **Preferences** dialog box
- **debuggerdirectives**
Selects or clears the [Use !debugger directives](#)^[68] checkbox in the **Preferences** dialog box
- **historyToTrace**
Selects or clears the [History To Execution Trace](#)^[134] checkbox in the **Preferences** dialog box
- **ieAuto**
Sets the [IE Mode](#)^[383] option in the **Preferences** dialog box to **proxy** or **none**
- **ignoredFileTypeList**
Toggles Debugger [file-type filtering](#)^[376] on or off:
setPreference ignoredFileTypeList {on|off}
Where:
 - **on** honors the "ignore-list" default behaviour
 - **off** does no file-type filtering
- **macroAutorun**
Selects or clears the [Macro Autorun](#)^[318] checkbox in the **Preferences** dialog box
- **pauseAtEndEval**
Selects or clears the [Pause at end of evaluation](#)^[61] checkbox in the **Preferences** dialog box
- **useProcLists**
Selects or clears the [Use Proc Lists for exclude/include](#)^[70] checkbox in the Client's **Proc Selection** tab
- **useRoutineLists**
Selects or clears the [Use Routine Lists for exclude/include](#)^[70] checkbox in the Client's **Proc Selection** tab

- `valueDisplayOnConsole`

Controls whether [value displays](#)^[101] appear in a separate **Value** window when the [Console](#)^[316] is open (the default, **on**, shows them in the **Console** if it is open)

Note: Instead of **on** or **off**, the command also takes **1** or **0**.

You may want to use the command to toggle a setting in a Client [macro](#)^[309], for example:

```
# toggle use exclude/include directives
toggle &debuggerDirectives !Directives
setPreference debuggerDirectives &debuggerDirectives
```

Client menu —

Introduced Build 55

5.122 **setStatusMessage** command

Action Lets you specify a message for display in the Client [Status bar](#)^[50].

Syntax:

```
setStatusMessage {&var/&&function/const}
```

where:

- *&var* is a macro variable that may or may not already exist.
- *&&function* is a macro function.
- *const* is a character string, which does not have to be enclosed in quotes.

Examples:

```
setStatusMessage Nothing happening!

set &msg = "Here's Johnny"
setStatusMessage &msg
```

Client menu —

Introduced Build 59

5.123 **setTitle** command

Action

Lets you change the title of the Client main window (which by default is "The Janus Debugger" or "The Sirius Debugger.")

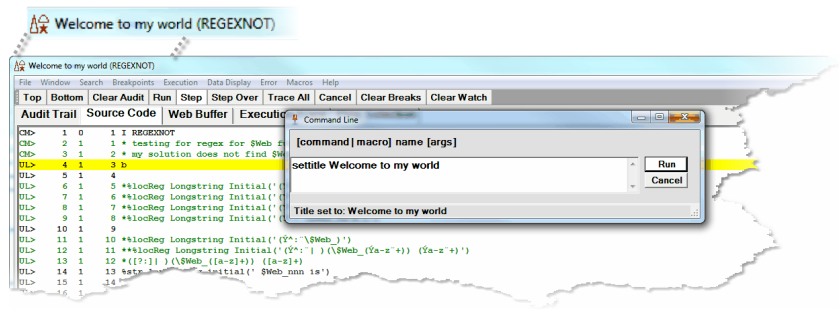
Syntax:

```
setTitle newtitle
```

where *newtitle* is a non-quoted, case-insensitive string of as many as 50 characters.

Examples:

- The image below shows the result of issuing a **setTitle** **Welcome to my world** command. Notice that the name of the procedure being debugged is retained.



- The following macro uses **setTitle**:

```
continueMacroIf &&blackOrWhiteList <> 'black'  
labelButton button0 BlackList Off  
turnOnBlackList  
setTitle Black List on  
set &changed = 1  
clearStatus
```

See also:

- The [restoreTitle](#)^[243] command replaces any changed main window title with the default title.
- [&¤tTitle](#)^[330] and [&&originalTitle](#)^[337] display the current and default titles of the Client main window.

Client menu

—

Introduced

Build 62

5.124 setWhiteList command

Action Lets you dynamically specify or clear a procedure [White List](#)^[79], which explicitly identifies the outer procedures you want the Debugger to debug. Non-listed procedures get executed, but their code is not sent to or displayed in the Client. This command temporarily overrides any existing White List, but it does **not** physically affect the contents of the white list file stored on disk (`whitelist.txt`). The command's effect is not persistent over runs of the Client.

Syntax:

```
setWhiteList [proc]...
```

where *proc* is one of possibly multiple, blank-separated, procedure file names. Name matching is case insensitive, and [wildcards](#)^[76] are allowed.

If no names are specified, the current white list is treated as empty. If one or more procedure names are specified, the contents of the white list are overridden by this new set of procedure names.

You might use this command for testing by putting it in a User Language procedure and sending that to the client via the existing [SIRIUS DEBUG CLIENTCOMMAND](#)^[155] *Model 204* command. Doing this lets you have a mainframe-based testing harness that sets up a white list on the Client.

For example:

```
SIRIUS DEBUG CLIENTCOMMAND 'setWhiteList P.MOE P.LARRY'
SIRIUS DEBUG CLIENTCOMMAND 'turnOnWhiteList'
```

See also the [setBlackList](#)^[256] command.

Client menu —

Introduced Build 62

5.125 **showAbout** command

Action Displays the contents of the Client's "About box," which is accessed from the **About** option of the Client [Help menu](#)^[38].

Syntax:

showAbout

Displays the following in the **About** window:

- The *Sirius Mods* version with which this "build" of the Debugger Client is associated
- The number of this Client build
- A searchable summary of the features provided by this and past builds

Client menu **Help > About**

Introduced Build 43

5.126 **showCommands** command

Action Displays in alphabetical order and with simple definitions the entire set of Client [commands](#)^[179].

Syntax:

showCommands

Client menu **Help > Commands**

Introduced Build 50

5.127 showFunctions command

Action Displays in alphabetical order and with simple definitions the entire set of [Client functions](#)^[321].

Syntax:

showFunctions

Client menu Help > Functions

Introduced Build 58

5.128 showIE command

Action Displays the current Internet Explorer browser operating mode (the File menu > **Preferences** > **IE Mode** option or equivalent [setIEmode](#)^[258] command setting that is in effect), as well as the current values of IE settings that pertain to the Debugger Client.

Syntax:

showIE

Sample output:

```
Current ie mode: mergedPac
Current ie settings
...Proxy flags=(5)
...Proxy server=()
...Proxy bypass=()
...Proxy autoConfigUrl=(file://c:\debugger\stateFiles\debuggerInternalPac.js)
...Raw flag byte=(05)
```

Common values for **Proxy flags** (in the output above) are:

- 5 A proxy automatic-configuration (PAC) file is being used
- 3 A proxy server is being used
- 1 A direct connection to the Internet is being used (no proxy or automatic-configuration file)

Client menu —

Introduced Build 62

5.129 **showShortcuts** command

Action Displays the Client's current ([default](#)^[291]) as well as mapped) keyboard shortcuts.

Syntax:

showShortcuts

Client menu **Help > Keyboard Shortcuts**

Introduced Build 43

5.130 **skipPreview** command

Action When the [Source Preview feature](#)^[85] is enabled, runs the program code normally but without downloading the source code for viewing or controlled execution.

Syntax:

skipPreview

Client menu **Execution > Skip Previewed Source**

Introduced Build 30

5.131 **span command**

Action Restores to the default the spanning behavior of a Debugger [macro](#)^[309], reversing any previously issued [noSpan](#)^[229] command. By default, macro execution spans the evaluation of requests (if a request completes before a macro is finished, the remaining macro commands apply to the next request).

Syntax:

span

Scope Allowed only in Debugger macros; **not** available as a mappable Client command

Client menu —

Introduced Build 37

5.132 **step command**

Action [Executes a single statement](#)^[55] in the code displayed in the Client's **Source Code** or **Daemon** page.

Syntax:

step

Client menu **Execution > Step**

Introduced —

5.133 **stepOut** command

Action [Steps out](#)^[64] of a subroutine or method.

Syntax:

stepOut

Client menu **Execution > Step Out**

Introduced —

5.134 **stepOver** command

Action [Executes a single statement](#)^[55] in the code displayed in the **Source Code** or **Daemon** page, but bypasses a subroutine or method.

Syntax:

stepOver

Client menu **Execution > Step Over**

Introduced —

5.135 toggle command

Action

Reverses the state ("True" or "False") of a [macro variable](#)^[319]. If the variable value is `0` or a zero length (null) string, or if it is undefined, its state is considered to be `False`. For all other values, its state is considered to be `True`.

The `toggle` command sets to 0 ("toggles to False") a variable whose state is True. And it sets to 1 ("toggles to True") a variable whose state is False.

Syntax:

```
toggle &var [msg_string]
```

where:

- `&var` is a macro variable that may or may not already exist.
- `msg_string` is an optional message string, useful for constructing a message to report what was toggled.

If `msg_string` is specified, either of the following is displayed in the [Status bar](#)^[50] after the toggle:

`"msg_string is on"` (if toggled on)

`"msg_string is off"` (if toggled off)

Example:

```
# toggle history  
toggle &historyWanted Auto History
```

Client menu —

Introduced Build 37

5.136 **toggleBreakpointOnCurrentLine** command

Action

Sets (or removes) a [breakpoint](#)^[57] for the current **Source Code** or **Daemon** line, if the line is or starts an executable statement. If the line is not an executable statement or the start of one, sets (or removes) a breakpoint for the first subsequent line that is or starts an executable statement.

For code lines for which a breakpoint is set, **toggleBreakpointOn** removes the breakpoint.

Syntax:

toggleBreakpointOnCurrentLine

This command has the same effect as double-clicking a code line or right-clicking a line and selecting **Toggle Breakpoint** from the context menu.

toggleBreakpointOnCurrentLine always sets or clears a breakpoint; [setBreakpointOnCurrentLine](#)^[257] performs only a set action, and [clearBreakpointOnCurrentLine](#)^[188] performs only a clear.

See also the [breaksAt](#)^[185] and [breaks](#)^[184] commands.

Client menu

Breakpoints > Toggle Breakpoint on Current Line

Introduced

Build 28

5.137 toggleInitExclude command

Action Inverts the way [Exclude mode](#)^[67] operates so that it initially excludes code instead of initially including code (until an explicit directive).

Syntax:

toggleInitExclude

Requires at least version 7.6 of the *Sirius Mods*.

Client menu Execution > Toggle Init Exclude

Introduced Build 47

5.138 toggleLower command

Action Changes the display of the Client's main window either to hide the [lower section](#)^[14] (if currently it is not hidden) or to restore the display of the lower section (if currently it is hidden).

This command is equivalent to either selecting or clearing the **Hide Lower Section** option of the **Main Window Options** section of the [Preferences](#)^[18] dialog box.

Syntax:

toggleLower

toggleLower always hides or restores the lower windows; [hideLower](#)^[213] performs only a "hide" action, and [restoreLower](#)^[243] performs only a "restore."

Client menu File > Preferences > (Main Window Options) Hide Lower Section

Introduced Build 57

5.139 top command

Action Scrolls to the top of the currently displayed page; has no effect if the **Proc Selection** page is being displayed.

Optionally, scrolls to the top of *the Client window you specify* as the value of the command's **In window** prefix.

Syntax:

[In window] top

where *window* is one of the following keywords, not case sensitive, which identify a Client window. Links are provided to help identify the less common windows:

[about](#)^[39]
auditTrail
[commands](#)^[38]
console
[executionHistory](#)^[134]
executionTrace
[keyboardShortcuts](#)^[38]
source
[textviewer](#)^[149]
[value](#)^[101]
watchWindow
webBuffer

Client menu **Window > Top**

Introduced —

5.140 trace command

Action [Traces](#)^[130] execution, and records executed lines.

Syntax:

trace

Client menu **Execution > Trace To End**

Introduced Build 26

5.141 **traceUntilVariableEqualsValue** command

Action Steps through the program being debugged, stopping if a statement modifies the value of the variable specified in the input box so that it equals a value you specify. Displays in the **Execution Trace** tab the statement that modified the variable and the new variable value.

Same as [Alt key + Run to Change](#)^[134] button.

In a macro, the command syntax is:

traceUntilVariableEqualsValue *variable target*

where:

- *variable* is the required variable name
- *target* is the required target *variable* value

Client menu Execution > Trace Until Variable Equals Value

Introduced Build 26

5.142 **traceValues** command

Action Same as [run](#)^[244] command, but also reports all statements that modify the value of the variable specified in the text box, and reports what value was assigned to the variable. Described further in [Tracing all updates to a variable's value.](#)^[132]

In a macro, the command syntax is:

traceUntilValues *varName*

where *varName* is the required variable name.

Client menu Execution > Trace Values

Introduced Build 26

5.143 **turnOffBlackList** command

Action Deactivates [Black List filtering](#)^[81] (like clicking the **Turn off Lists** button on the **Proc Selection** page).

Syntax:

turnOffBlackList

Client menu **Execution > Turn Off Black List**

Introduced Build 62

5.144 **turnOffDebugging** command

Action Turns off Debugger processing (like executing [SIRIUS DEBUG OFF](#)^[151]).

Syntax:

turnOffDebugging

Client menu **Execution > Turn Off Debugging**

Introduced —

5.145 **turnOffWhiteList** command

Action Deactivates [White List filtering](#)^[81] (like clicking the **Turn off Lists** button on the **Proc Selection** page).

Syntax:

turnOffWhiteList

Client menu **Execution > Turn Off White List**

Introduced Build 28

5.146 **turnOnBlackList** command

Action Activates [Black List filtering](#)^[81] (like clicking the Turn On Black List button on the Proc Selection page).

Syntax:

turnOnBlackList

Client menu Execution > Turn On Black List

Introduced Build 62

5.147 **turnOnWhiteList** command

Action Activates [White List filtering](#)^[81] (like clicking the Turn On White List button on the Proc Selection page).

Syntax:

turnOnWhiteList

Client menu Execution > Turn On White List

Introduced Build 28

5.148 unPin command

Action Unpins the specified Client [external window\(s\)](#)^[300], that is, no longer forces the window(s) to remain at the top of the Client PC's open window stack.

Syntax:

```
unpin {windowname | pattern | *}
```

where you must specify one of these:

- *windowname*, the (case not important) name, or title, at the top of a Client work window or external window
- *pattern*, a character sequence that ends with an asterisk (*), which performs a "wildcard" search (for example, *w** finds the **Watch Window** and **Web Buffer** window)
- A lone asterisk (*), which removes the pinning from all pinned windows

If you **unPin** a window that is not already pinned, the command is ignored. If the command indicates a window that is missing or invalid, or if it does not match an open external window, an error message is issued.

To pin a window that is not currently pinned, use the [pin](#)^[236] command.

Client menu Context menu option (**UnPin**) of external window title bar

Introduced Build 56

5.149 unSet command

Action Lets you remove a [macro variable](#)^[319].

Syntax:

unSet &*target*

where &*target* is the macro variable being removed.

Note: To unset a macro variable or macro function in a Client [mapping command](#)^[288], you need to XML entity-encode each ampersand (&). For example, to map the command **unset** &foo, you specify it like this:

```
<mapping command="unset &amp;foo" button="button14"/>
```

Client menu —

Introduced Build 59

5.150 valueDisplay command

Action Acts like the [Value button](#)^[101], displaying in a separate window the value of the item specified or the value currently in the [Entity-name input box](#)^[51].

Syntax:

valueDisplay [*value*]

where *value* is the program item whose value you want to display.

If *value* is not specified, the command attempts to use the value specified in the Entity-name input box. For example, you might want to map a button you can readily click to display the value of what is in the Entity-name input box:

```
mapbutton Button0 valueDisplay
```

The **Value** window opened by **valueDisplay** can be closed by [closeValueDisplay.](#)^[196]

In a macro, the variable name is specified as an argument, for example: **valueDisplay %i**

If `valueDisplay` is executed, and the [Console](#)^[316] is open, the value is displayed in the **Console** window. To insist that a Value window be used for the display in this case, use the `valueDisplayOnConsole` option of the [setPreference](#)^[260] command.

Client menu **Data Display > Value Display**

Introduced Build 43

5.151 **varDump** command

Action Displays the current values of all [macro variables](#)^[314] defined during this Client session. The message is normally displayed in a standard Windows informational box (entitled **varDump for Macro**). If the [macro console](#)^[316] is open, however, the message is sent to the console instead.

Syntax:

varDump

The command output is a display of the count of the number of macro variables, followed by an alphabetically ordered list of the individual variables (including [&argstring](#)^[314]) in name="value" format. For example:

```
8 macro variables
&a1="1"
&a2="-1"
&a3="666"
&a4="4"
&a5="5"
&a6="All work and no play makes jack a dull boy..."
&argstring=""
&b="1"
```

Client menu —

Introduced Build 28

5.152 **viewText** command

Action Invokes a [separate viewer](#)^[149] for copying, printing, and saving Client text data.

Syntax:

viewText

Client menu Window > View Text

Introduced —

5.153 **windowToTop** command

Action Brings the specified Client window to the top of your screen's stack of application-windows. For example, specifying **windowToTop** **Web Buffer** brings an [external](#)^[300] **Web Buffer** window to the foreground of your monitor screen.

Syntax:

windowToTop *windowname*

where *windowname* is the (case not important) name, or title, at the top of a Client work window or external window. *windowname* may end with an asterisk (*) to perform a "wildcard" search (for example, **ab*** finds the **About** window).

Client menu —

Introduced Build 50

CHAPTER 6 *Customizing Client Operations*

The Debugger Client standard operations are mainly controlled by a default set of buttons and hot keys and their menu-option counterparts. Those controls activate a set of underlying commands to which you have access. This chapter describes how you can create alternative mappings of these commands to reassign the operations that the buttons and keys perform.

The chapter also describes the options available for changing the default colors of the text or background of the various Client displays, how to change the location of the files the Client creates and uses during debugging sessions, and a variety of options for detaching and arranging the Client's constituent windows.

Note: In addition to the customizations described in this chapter, the Client's **Preferences** window (accessed via the **File** menu) contains multiple [options](#)^[18] for controlling the operation of the Client.

The sections included in this chapter are:

[Reconfiguring GUI buttons and hot keys](#)^[283]

[Changing the colors in Client displays](#)^[293]

[Changing the location of Client work files](#)^[296]

[Changing the font size in Client displays](#)^[299]

[Opening an external window](#)^[300]

[Hiding the Client's lower windows](#)^[305]

[Seeing through Client windows](#)^[307]

6.1 **Reconfiguring GUI buttons and hot keys**

The fifteen [buttons](#)^[285] in the [main button bar](#)^[39] (and those in any [extra button bar](#)^[43]) are fully configurable. You can set the buttons to perform commands or macros, and you can rearrange their positions or add [separator buttons](#)^[285].

You may also create your own [keyboard shortcuts](#)^[286] (also called "hot keys") for commonly used commands.

If necessary, you can even provide an alternative customization that overrides the primary reconfiguration. This lets you provide, say, group-level settings that may be overridden by individual-level settings.

The default setting of a configurable button is provided where the individual button is discussed in this document, and the settings are also summarized [later](#)^[291] in this section.

To reconfigure buttons or keyboard shortcuts, you can define a mapping file or use a Client command.

Using a mapping file

1. [Create](#)^[287] an XML file that contains the [command](#)^[179] or [macro](#)^[309] (multiple commands) to execute paired with the button and/or key that invokes it.

This file must be named `ui.xml` and [is assumed](#)^[296] to reside in the installation target folder (along with the JanusDebugger.exe Client program).

2. Restart the Debugger Client.

Using a Client command

As of Client Build 56, there is an alternative way to reconfigure buttons or keyboard shortcuts that does not involve a mapping file or a Client restart:

1. From the Client's [Command Line](#)^[317] tool, or within a [macro](#)^[309], or from within a User Language request by using the ClientCommand method of the [DebuggerTools](#)^[161] class, issue these commands as necessary:
 - [mapButton](#)^[224], which specifies the button [mapping](#)^[288] you want
 - [mapKey](#)^[224], which specifies the keyboard shortcut you want
 - [clearButton](#)^[189], which removes the button mapping you specify
 - [clearKey](#)^[191], which removes the keyboard shortcut you specify
2. View the button arrangement or test the key combination you specified or removed.

Unless you changed only a button modifier (a key you press along with clicking the button), an updated button's new label is displayed and the button is available for immediate use. A cleared button is removed immediately from its button bar.

The button-mapping and hot-key updates you make with `mapButton` or `mapKey` are **not** reflected in the `ui.xml` file, and they do not persist through subsequent Client sessions.

Additional Client commands let you modify buttons:

- [disableButton](#)^[202] and [enableButton](#)^[204] disable and enable button bar buttons.
- [labelButton](#)^[219] lets you replace the label of a button.

In this section

The following subsections discuss the eligible commands, buttons, and keys, how you create a `ui.xml` file for their reconfigurations, and what their default settings are:

[Introducing the configurable components](#)^[285]

[Setting up the ui.xml file](#)^[287]

[Default settings of buttons and hot keys](#)^[291]

6.1.1 Introducing the configurable components

The activities you invoke from the Debugger Client GUI are also available as commands you can assign to different Client buttons or keyboard keys or combinations of both of these, as well as to Debugger [macros](#)^[309]. This section describes the commands, buttons, and keys you can associate [in a mapping file](#)^[287] or [using a mapping command](#)^[284] to reconfigure the Client user interface.

Commands

Client commands are the operations that you invoke from Client menus and can assign to a Debugger Client button, keyboard shortcut, or macro. The act of associating a button or keyboard shortcut with a command or macro (multiple commands) is termed "mapping."

[The Client command reference](#)^[179] describes the available commands. The default hot keys and buttons with which some of these commands are associated are summarized [later](#).^[291]

Named buttons and separators

When mapping a command or macro to a button in the `ui.xml` file, you refer to the button by name:

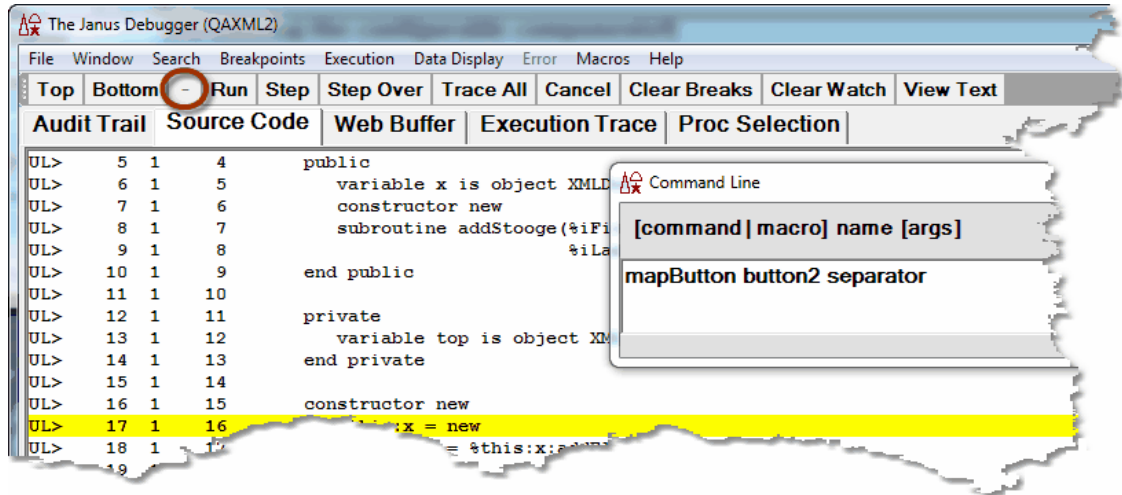
- From left to right, the fifteen mappable buttons in the [main button bar](#)^[39] are named `button0`, `button1`, ... `button14`. The buttons have initial [default mappings](#).^[291]
- The fifteen mappable buttons in the [extra button bar](#)^[43] are named `extrabutton0`, `extrabutton1`, ... `extrabutton14`. These buttons have **no** initial default mappings.

Only button bar buttons are mappable. No other Client buttons are mappable.

An additional option (as of Client build 57) is to map a named button to a special command keyword (**separator**) that converts the button to a **separator**, a button whose sole function is to provide a visual separation between buttons, say for better visual discrimination or to create button groupings. Such a button is always dimmed and not responsive.

The **separator** keyword is available for the [mapButton](#)^[224] command and the mapping-file [mapping](#)^[288] element.

The following image shows a newly created separator button:



Button modifiers

Normally, when you map a command or macro to a button, the command or macro is executed when you click the button. However, you can also map a command or macro to a "modified" button: the command or macro executes only when you click a button while you are holding down the Alt or Ctrl key on your keyboard.

This mechanism lets you map multiple commands or macros to one button. The usual click of the button executes one command or macro, while pressing Alt and clicking the button executes a second command or macro, and pressing Ctrl and clicking the button executes a third command or macro.

A button's label is not affected by specifying a modifier for the button. The label always indicates the primary, unmodified, function of the button.

Keyboard shortcuts

Often, a keyboard shortcut is the best way to do an important operation. You may map a keyboard shortcut to any alphabetic key (lowercase and uppercase are differentiated), numeric key, or to any of the function keys except F1 (that is, F2, F3, . . . F12). As with mapping buttons, you can specify an Alt or Ctrl modifier key for a keyboard shortcut.

To be reminded of the current ([default](#)^[291] as well as mapped) keyboard shortcuts, select **Keyboard Shortcuts** from the **Help** menu.

6.1.2 Setting up the ui.xml file

Each time the Debugger Client is started, it checks for the presence of a **ui.xml** file in the same folder as the Client executable file itself (the default) or [in the folder specified in the Client configuration file](#)^[296]. If no such file is located, the Client opens with its default presentation of GUI buttons and their associated commands, and it responds to its default set of hot keys and their associated commands.

If you want to change the Client's [default buttons and hot key associations](#)^[291], you must provide in the appropriate folder a **ui.xml** text file that adheres to the structure described in this section.

Note: You can also define an alternative to the **ui.xml** file: following the same rules for setting up the **ui.xml** file, you provide settings in the [uimore.xml file](#)^[291] that override those in the **ui.xml** file.

And/or:

You can [use the mapButton or the mapKey command](#)^[284] to reconfigure buttons or hot keys without editing a mapping file.

To set up a **ui.xml** file:

1. From the Client's **File** menu, select **Edit ui.xml**.

An untitled *Notepad* file is opened for you, along with a prompt to create a **ui.xml** file. (If a **ui.xml** file already exists, that file is opened.)

2. In the file, specify or update the **mappings** tag, the first line in the file.

The top level tag in the **ui.xml** file is the **mappings** tag, which has two optional attributes (**useDefaults** and **startUpMacro**):

```
<mappings [useDefaults="true|false"] [startUpMacro="macroname"]>
```

useDefaults indicates the mode in which the mappings are applied:

- **useDefaults="true"** tells the Client to apply the ui.xml file mappings **after** the defaults are set for the buttons and hot keys. In this mode, your mappings are additions or overrides to the existing defaults.

Use this mode if you generally like the defaults but want to make a small number of additions or changes to them.

- `useDefaults="false"` tells the Client *not* to set the defaults, just to process the `ui.xml` file mappings. You start with a "clean slate" and only the settable buttons and hot keys you assign will be available.

Use this mode if you want to completely change the settings from the defaults.

This is the default.

- `startUpMacro` indicates a [macro](#)^[309] that is run when the Debugger Client starts.

3. Specify or update `mapping` sub-elements for your buttons/keys.

Contained within the `mappings` element are one or more `mapping` sub-elements. Each `mapping` element associates a command or macro with a button, a hot key sequence, or both. If it associates a command or macro with a button, it may also specify a modifier for the button.

The `mapping` element attributes are described below, after which is a comprehensive example:

`command`

A double-quoted string that contains a Client command, each of which is described in the [command reference](#)^[179], and two of which deserve special mention here:

- The [macro](#)^[221] command calls a user-defined macro
- The `separator` keyword creates an inactive button whose only function is to be a visual [button separator](#)^[285] on a Client button bar

The `command` attribute is required.

The quoted command string may include a command parameter or an "In *window*" prefix (as of Client build 57) if appropriate. Check the individual command description in the command reference. As an example:

```
<mapping command="runUntilVariableChanges %i"
  button="button12"/>
```

Note: If a command parameter contains an ampersand (&), you need to XML entity-encode it. For example, to map the command `set &foo = %i`, you specify it like this:

```
<mapping command="set &foo = %i"
  button="button14"/>
```

`button`

A [named button](#)^[285] (`button0`, . . . , `button14` for the main [button bar](#)^[39], and `extrabutton0`, . . . , `extrabutton14` for an [extra](#)^[43] button bar).

If no `button` setting is specified, a `key` setting must be specified.

`buttonModifier`

A [button modifier](#)^[286] (Alt or Ctrl). This attribute is optional.

key One of these keyboard keys: a-z, A-Z, or F2-F12. If no **key** setting is specified, a **button** setting must be specified.

keyModifier A [key modifier](#)^[286] (Alt or Ctrl). This attribute is optional.

Note: As with all XML, the element and attribute names are case sensitive. Attribute values, which must be quoted and non-null, are *not* case sensitive.

For any given **mapping** element, you must specify a **button** or **key** setting (or both). Here is an example:

```
<mappings useDefaults="true" startUpMacro="myDefault">
  <mapping command="step" button="button0" />
  <mapping command="run" button="button1" key="f5" />
  <mapping command="cancel" button="button2" />
  <mapping command="stepOut" button="button3" />
  <mapping command="runWithoutDaemons" button="button4" />
  <mapping command="in audittrail top" button="button5" />
  <mapping command="in audittrail bottom" button="button5" buttonModifier="alt" />
  <mapping command="traceValues" button="button6" />
  <mapping command="breakOnNextProc" key="9" />
  <mapping command="nextCompileError" key="F2" />
  <mapping command="previousCompileError" key="F2" keyModifier="alt" />
  <mapping command="macro stepstep" key="Z" keyModifier="ctrl" />
  <mapping command="openMacroConsole" button="button10" />
  <mapping command="closeMacroConsole" button="button11" />
  <mapping command="showCommands" button="button12" />
  <mapping command="help" button="button13" />
  <mapping command="step" button="button14" />
  <mapping command="manual" button="extrabutton0" />
  <mapping command="showAbout" button="extrabutton1" />
  <mapping command="showShortcuts" button="extrabutton2" />
  <mapping command="manual" button="extrabutton3" />
</mappings>
```

If you define more than one mapping for the same key or button, the last mapping (closest to the `</mappings>` end tag) takes precedence.

If you specify a macro, it is assumed to reside in the same folder as the Debugger Client executable file unless you [configured a different location](#).^[296]

4. Save, then Exit the file.

If you edited an existing `ui.xml` file, the updated settings take effect the next time the Client is started.

Button toggle for compilation errors

The following two commands are only meaningful in the context of [examining the results of a failed compilation](#)^[138]. If the program you are debugging has no compilation errors, these commands do nothing:

previousCompileError
nextCompileError

Since a button you map to one of these commands is not likely to often be used, you are allowed to map these commands (and only these) to a button that has another command or macro mapped to it. Then, if a request you are debugging encounters a compilation error, the button switches its association from the first command or macro you specified to the compile error function.

The button toggle occurs only when viewing a failed compilation — in a non-error context, repeated button clicks execute only the first command specified for that button.

Here is an example:

```
<mapping command="stepOver" button="button8"/>
<mapping command="nextCompileError" button="button8"/>
<mapping command="stepOut" button="button9" />
<mapping command="previousCompileError" button="button9" />
```

Validation of mappings

When you (re)start the Client, the ui.xml file is validated for XML structure and conformance to the rules given above for specifying its elements and attributes. Parsing violations produce an error message, the Debugger Client opens, and either of the following result, depending on whether the offending attribute was optional or required:

- The offending attribute is absent, but the other items are as mapped.
- All the mappings are rejected, and the Client has a set of default mappings.

For semantic violations like misspelling a command name or specifying a command that is not supported in the current Client build, the Client opens with an error message, and the offending attribute is absent while the other items are as mapped.

Note: You may set a command to a button with a modifier only if you have set a command for that button without a modifier.

In other words, to successfully map a command to an "Alt+button" or "Ctrl+button" combination, you must first map the command to an unaccompanied button click.

Overriding the ui.xml file

It may be suitable at your site to provide a second level of overrides to the default button and key mappings of the Debugger Client. For example, you might define a `ui.xml` file that contains a set of standard mappings for all the users in a group, and let individuals in the group override the group settings by defining their own mappings in a `uimore.xml` file.

To override the `ui.xml` file:

1. Define a `ui.xml` file following the guidelines specified [above](#).^[287]
2. In the same file folder and using the same format, define a `uimore.xml` file.

You can use the editing tool of your choice, or you can use the **Edit uimore.xml** option in the Client's **File** menu (as of Client build 57).

The settings in the `uimore.xml` file override those in the `ui.xml` file, and they will be subject to the same [structure and content validation](#).^[290]

3. [Restart](#).^[18] the Debugger Client.

The `uimore.xml` file is read **after and only if** a valid `ui.xml` file is read. If no `ui.xml` file is present or if it contains a significant error, the `uimore.xml` file is not processed.

6.1.3 Default settings of buttons and hot keys

Unless you have [provided](#).^[287] a user interface reconfiguration file, the Debugger Client starts with the default presentation of GUI buttons and hot keys summarized in this section.

Buttons

The ten buttons above the Client's main window tabs are initially set to perform the commands that are shown in the table below (and described further in [The Client command reference](#)).^[179] The buttons are named by their left-to-right position. The button labels shown in the Client are close approximations if not the same as the name of the command they execute, and the labels are configurable with the `labelButton`.^[219] command as of Build 62. Five additional buttons are available for mapping but have no command associated with them by default, and they do not appear unless they are explicitly mapped:

Button	Default command setting
button0	top
button1	bottom
button2	clearAudit
button3	run
Alt+button3	runWithoutDaemons
button4	step
button5	stepOver
Alt+button5	stepOut
button6	trace
button7	cancel
button8	clearBreaks
button9	clearWatch
button10	— (none)
button11	— (none)
button12	— (none)
button13	— (none)
button14	— (none)

Hot keys

The Client's default set of hot keys and the commands that they perform are shown below.

Hot key	Default setting
Alt+B	breaksAt

Hot key	Default setting
Ctrl+B	breaks
Ctrl+C	copy
Ctrl+F	focusToSearchBox
Ctrl+P	preferences
Ctrl+T	trace
Ctrl+U	searchFromBottom
Ctrl+X	cancel
F4	step
F5	run
Alt+F5	runWithoutDaemons
F9	searchDown
Alt-F9	searchUp
F10	stepOver; also, previousCompileError
Alt+F10	stepOut
F11	step; also nextCompileError

Note: By default, the Enter key repeats the command performed by the currently active button (which is highlighted with a white background).

6.2 Changing the colors in Client displays

Whether to boost visual discrimination or simply to add visual variety, you can change the color of the text and backgrounds in the various Client windows and pages. You may want to make your code comments stand out by displaying them in a different color, for example. Or maybe you want to change the highlighting the Client uses for breakpoints because your eyes are insensitive to the default maroon.

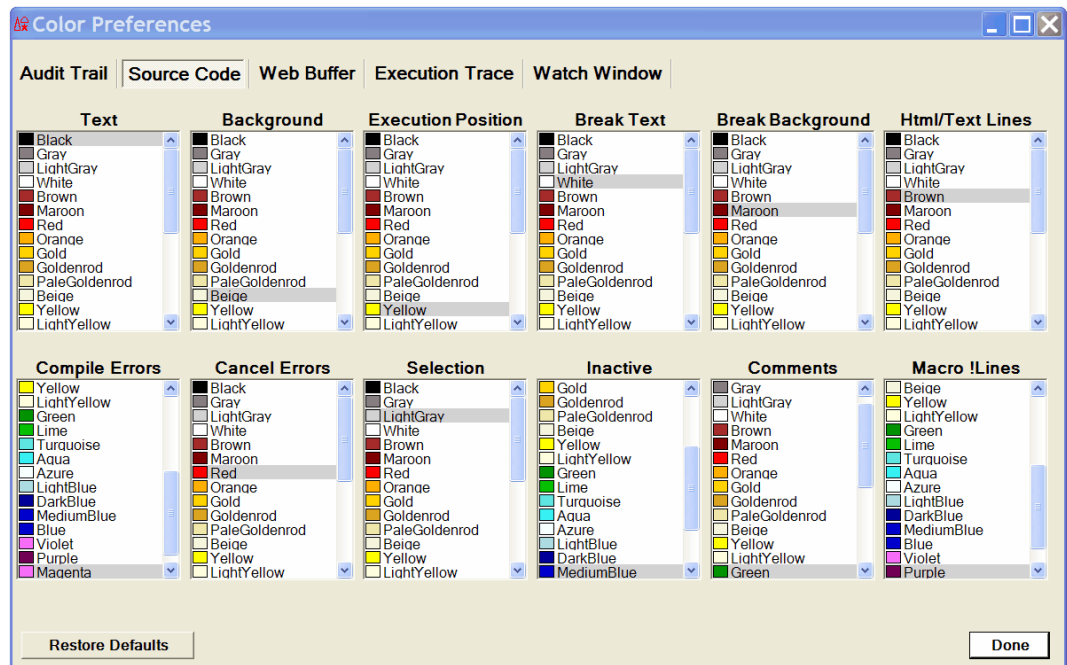
The **Color Preferences** option in the Client's **File** menu leads to a set of panels, each of which represents an element on the Client whose color can be modified. The panels display the (standard, Windows) colors you can apply to the Client element.

When you select a color, it is immediately displayed on the specified Client window or page. You can test as many colors and elements as you want, and you can return at any time to the default Client color settings. Any color modifications you make remain through subsequent sessions of the Client.

To make a color change

1. In the Client's **File** menu, select **Color Preferences**.

The **Color Preferences** window displays the color options for the current Client page or window, highlighting with a gray background stripe the default or existing color values:



The window contains separate tabs (five) for the Client areas for which you can make color changes. Each tab contains a labeled panel for each display element whose color you can change.

The **Audit Trail** tab controls both the Client **Audit Trail** page and **Most Recent Audit Trail** window. The Client **Proc Selection** page has no color-changeable elements, and the Client **Daemon** page colors are the same as those for the **Source Code** page.

2. Select the tab for the Client area you want to modify.

The panels you see are those that are available for the tab you selected. The labels above the panels indicate the type of element to which the color applies. These elements are described further in the table below.

3. In the panel that represents the display element you want to color, click anywhere in the row of the color you want for this element.

The color is immediately reflected in the Client, and it persists through Client sessions unless you change it again.

4. Experiment further with other color changes, or click the **Done** button to exit. Click the **Restore Defaults** button to return to the original Client color values.

The color panels

The following table describes the Client elements that are represented by the **Color Preferences** color panels.

Panel label	Page or window	Refers to
Text	all	All text in the page or window. The color is subject to override if it is for "inactive" code lines shown in the Source Code page.
Background	all	The background color of the page or window.
Selection	all	The last reported, or last user-selected, display line.
Execution Position	Source Code	The line of code that is next to execute.
Break Text	Source Code	The text in a breakpoint ^[57] line.
Break Background	Source Code	The background color of the row that contains a breakpoint line.
Html/Text Lines	Source Code	The text within User Language HTML and TEXT statement blocks.
Compile Errors	Source Code	The text of compilation error messages.
Cancel Errors	Source Code	The background color of the row that contains a line of code whose execution produced a request cancellation.
Inactive	Source Code	The text color of code that is currently not executable or able to respond while the executable code in a Client Daemon page is active.

Panel label	Page or window	Refers to
Comments	Source Code	<p>The text in User Language code comment lines (any that begin with an asterisk).</p> <p>Note: You must be running at least version 7.6 of the <i>Sirius Mods</i> to add color to in-line or multi-line comments (bounded by characters like <i>/?</i> and <i>?/</i> defined by <i>Model 204</i> COMSTART and COMEND commands).</p>
Macro !Lines	Source Code	The text in User Language Macro Facility statements. These statements begin with an exclamation character (!).
Most Recent	Web Buffer	The background color of the lines most recently added to the page.
Changed Values	Watch Window	The background color of the row that contains a watched item whose value is changed by the last statement execution.
Out of Scope	Watch Window	The text color of a row that contains (the last value of) a watched item that is no longer in scope, that is, not within code in the currently active tab (Source Code or Daemon).

6.3 Changing the location of Client work files

The Debugger Client uses a variety of text files to store information about user activity to maintain its tools and displays, as well as a comprehensive troubleshooting log. These work files are catalogued [below](#)^[297].

The Client [configuration file](#)^[374] (`debuggerConfig.xml`) has optional XML elements with which you can specify alternative folder locations for the Client work files. The files are divided into three groups, each of which is controlled by a single element, as follows:

- `<stateFileFolder>` specifies where most Client work files are written (log, preferences, searches, for example)
- `<uiFolder>` specifies where the Client interface-customization files (ui.xml, uimore.xml) are stored
- `<macroLibraryFolder>` specifies where Client macro files are stored

Within `debuggerConfig.xml`, you specify folder path values for the elements as in the following:

```
<debuggerConfig version="1.0">
    ...
    <stateFileFolder>c:\myData</stateFileFolder>
    <uiFolder>c:\myUI</uiFolder>
    <macroLibraryFolder>c:\work\macroLibrary</macroLibraryFolder>
    ...
</debuggerConfig>
```

If you include an element, the specified folder location is validated when the Client is started, and you receive an error if the folder is missing or not writable. The locations are also reported in the audit trail.

If an element is not specified, the installation target folder is the assumed location. The exception to this is the `macroLibraryFolder` element: if this element is *not* specified, the Client will initially attempt to store or find macro files in the `stateFileFolder` location, else in the Client installation folder. Similarly, if the `macroLibraryFolder` element *is* specified, the Client tries that location first, else it tries the `stateFileFolder` location, else the Client installation folder.

A summary of the Client work files

File (by type)	File content	Configuration file element that specifies file default location
.macro	Client macro	macroLibraryFolder
excludeProc.txt	The list of procedures to be excluded from debugging ^[67]	stateFileFolder
excludeRoutine.txt	The list of methods and subroutines to be excluded from debugging ^[67]	stateFileFolder
find.txt	The most recent Search ^[39] terms	stateFileFolder
includeProc.txt	The list of procedures not to be excluded from debugging ^[67]	stateFileFolder
includeRoutine.txt	The list of methods and subroutines not to be excluded from debugging ^[67]	stateFileFolder
log.txt	The Debugger Client log ^[348]	stateFileFolder

File (by type)	File content	Configuration file element that specifies file default location
until.txt	The most recent run-until ^[75] target procedures	stateFileFolder
vars.txt	The most recent entries in the text box above the Watch Window	stateFileFolder
watchmemory.txt	The contents of the Watch Window between Client runs (if feature ^[91] is enabled)	stateFileFolder
whitelist.txt	The white list ^[79] of procedures to omit from debugging	stateFileFolder
.watch	Saved ^[91] Watch Window lists	stateFileFolder
about.xml	The remembered position and size of the Client's About ^[38] box	stateFileFolder
console.xml	The remembered position and size of the Macro Console ^[316] window	stateFileFolder
getVariableList.xml	The remembered position and size of the getVariablesForClass ^[211] command output window	stateFileFolder
history.xml	The remembered position and size of the Execution History ^[134] window	stateFileFolder
preferences.xml	The settings from the Preferences ^[18] and Color Preferences ^[293] dialog boxes	stateFileFolder
shortcuts.xml	The remembered position and size of the Keyboard Shortcuts ^[286] window	stateFileFolder
textviewer.xml	The remembered position and size of the Text Viewer ^[149] window	stateFileFolder
ui.xml	Modifications ^[283] to the Client's default buttons and hot key associations	uiFolder
uimore.xml	Settings that override ^[287] the ui.xml file settings	uiFolder

File (by type)	File content	Configuration file element that specifies file default location
valueDisplay.xml	Remembered position and size of Value window displays ^[107]	stateFileFolder
windowmemory.xml	Remembered position and size of the main Debugger Client window	stateFileFolder

6.4 Changing the font size in Client displays

The Client [configuration file](#)^[374] (`debuggerConfig.xml`) has an optional XML element (**fontScale**) with which you can specify a larger font size than the default for Client displays. Your font scale changes are applied to the user data contained in these (tabbed, external, Value, console, history, watch, Help information) Client windows:

- Audit Trail, Web Buffer, Execution Trace** (tabbed or external)
- Source Code** (including **Daemon**) tabs
- Watch Window** (including external)
- Execution History** (tab or separate window)
- Console** (macro and command output)
- Value** (value displays, object and list expansions, records and field groups)
- Help** menu information options (**About, Keyboard Shortcuts, Commands**)

Note: Debugger scaling is applied after any scaling by Windows if you are running with a non-standard DPI.

To scale the font size

1. Specify the `fontScale` element at the root level in the `debuggerConfig.xml` file. For example:

```
<debuggerConfig version="1.0">
  <fontScale>2.5</fontScale>
  <serverList>
    ...
  </debuggerConfig>
```

The scaling value you specify may be from 1.00 to 9.99 (with 0, 1, or 2 decimal digits). The value is the factor by which the default apparent font size is multiplied and scaled accordingly. A value of 1 leaves the default font size unchanged; a value of 2 doubles it.

2. Save your change, and restart the Client.

If your `fontScale` value is valid and not 1, the Client Audit Trail display will resemble the following:

```
... 08:18:30    Executable: C:\debugger\ClientSource\bin\JanusDebugger.exe
... 08:18:30    Executable date: 9/27/2010 4:10:51 PM
... 08:18:30    Font Scale: 2.5 <<<<<<
```

6.5 Opening an external window

Although the Client's **Watch Window** is expandable (by dragging its left edge) and you can also view long values by hovering the mouse pointer, you still may find cases where you would like to have more room for the **Watch Window** than is available. Or you may have cases where you could take better advantage of your monitor screen real estate if the **Audit Trail** tab were separated from the Client. Or you may have dual monitor capability, and ideally would like to use one monitor for the Client main window and one monitor for the **Web Buffer** tab.

The "external window" feature of the Client lets you display any combination of the Client main areas (the **Watch Window**, **Audit Trail** tab, **Execution Trace** tab, **Web Buffer** tab) or the [button bar](#)^[39] in individual windows that are separate from each other and from the Client.

You invoke the feature by double-clicking the name of the Client area, or by menu option or mapped button, key, or macro. Any of these actions instantly creates a separate independent window for the specified Client area. The contents and label of the former area are removed and transferred to the new window, which (except for an external button bar) is equipped with Print and Save options accessible by menu or button. Closing the external window returns its contents to the usual area on the Client.

The rest of this section describes the feature with a detailed example, considers ways to control the positioning of external windows, and shows how to open an external window automatically.

Watch Window example

To use an external **Watch Window**, for example:

1. Do either of the following:

- a. Select **Open External Watch Window** from the Client's **Window** menu.

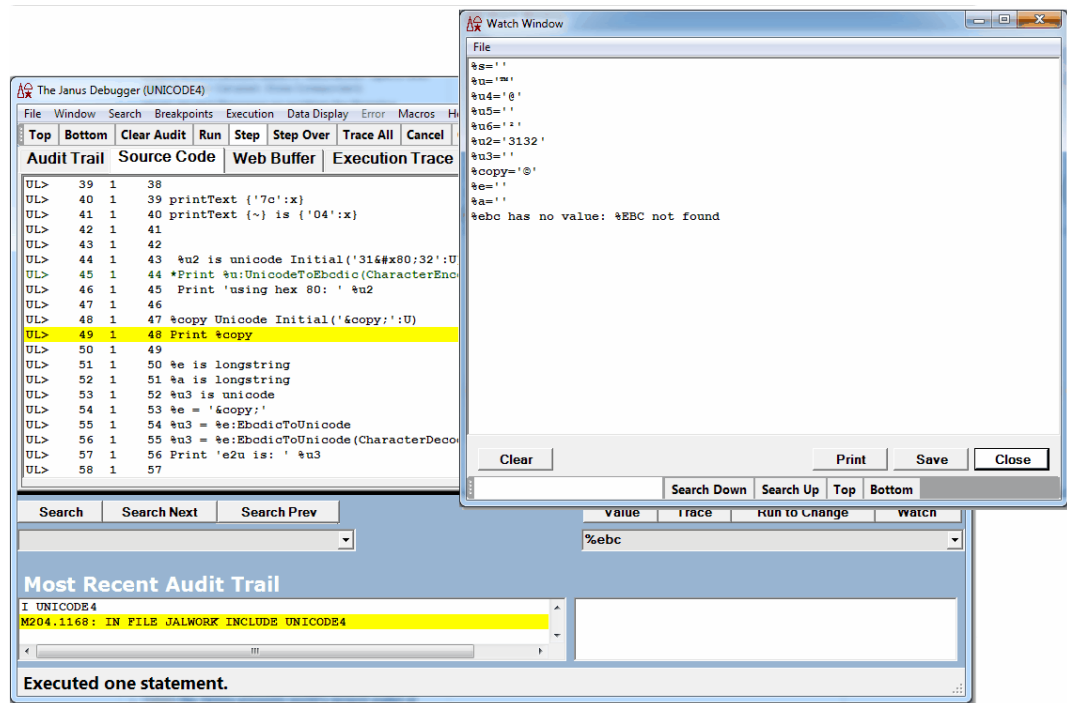
For opening the **Watch Window** (but not for the other areas) you can also use the **Data Display** menu.

- b. Invoke a button, key, or macro [mapped](#)^[283] to the [openExternalWatchWindow](#)^[233] command.

Unique open commands exist for each of the other areas.

As described [below](#)^[305], you may want to open the external window at Client startup.

2. The external **Watch Window** window opens, displaying the current **Watch Window** items. The space formerly occupied by the **Watch Window** is vacated, including its label:



In the external window:

- Print and save options are on buttons and in the **File** menu.
- The **Clear** button deletes all the items in the window.
- The **Close** button closes the external window but repopulates the former **Watch Window** area on the Client with the items that were present at the time of the close.

The mappable command [closeExternalWatchWindow](#)^[194] performs the same action as the **Close** button.

Unique close commands exist for each of the other areas except for the button bar, and the [closeExternalWindows](#)^[195] command closes multiple windows at once.

- A search bar provides a text box and control buttons for searching the window content. **Top** and **Bottom** buttons locate and highlight the first or last line of the content, while **Search Down** and **Search Up** activate backwards or forwards searching.

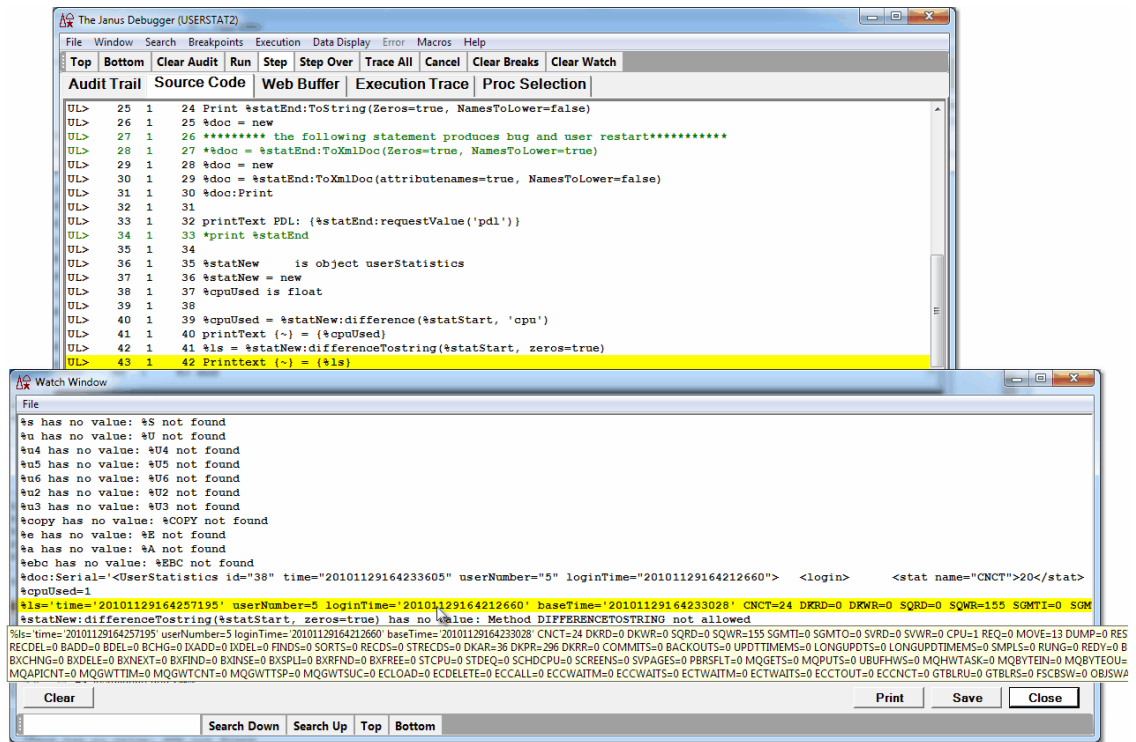
Pressing the Enter key after entering a search string in the search text box searches down, by default, and each subsequent key press searches for additional occurrences of the string, as long as the focus remains on the search text box.

3. Continue debugging. The external window functions the same as the normal one, except:
 - The external window typically pops up when you click it or when new content is written to it. Clicking a Client control brings the Client window to the top.
 - External windows may be [pinned](#)^[303]: you can specify one or more of them to keep in view at all times
 - Only the external window has print and remove-all-items-at-once (Clear) capabilities.

If you close and restart the Client, the external window location will remain what it was when you closed the Client.

You might want an external window simply to display more data, for example, data that is displayed in long lines. In the image below, which also shows the [show-as-tooltip feature](#)^[90], the very long data is more fully displayed in the external **Watch Window**.

Placing the external window below the Client also means both the Client and the external window remain in view as you step through the source code, even as variable value changes cause the external **Watch Window** to pop up in front of the Client window:

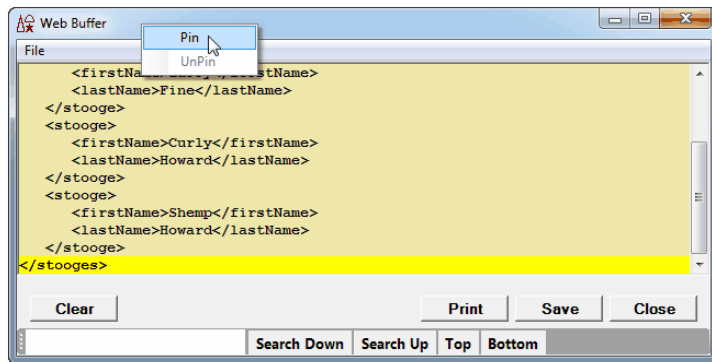


The following subsection describes an additional tool for controlling the viewing of multiple windows.

Keeping external window(s) constantly on top

With a proliferation of external windows on your PC desktop, it is sometimes useful to mark a window to remain at the top of the window stack (often called "pinning" a window). For example, you may want an external **Web Buffer** window to always be the "topmost" window, especially if you have large or multiple monitors.

As of Build 57 of the Debugger Client, you can "pin" an external window by simply right-clicking its title bar, then selecting the **Pin** option from the context menu:

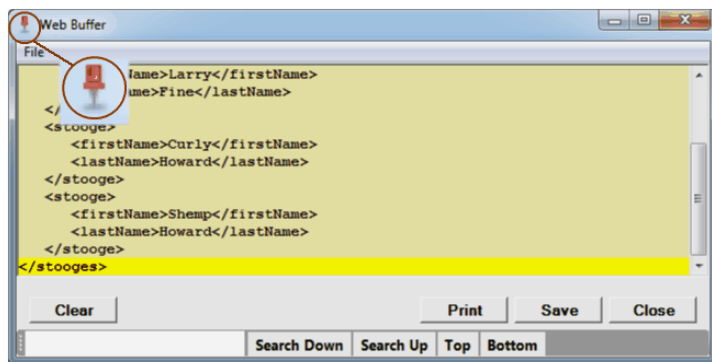


Alternatively, you can issue the Client [pin](#)^[236] command (Build 56). For example, to pin an external **Web Buffer** window, you issue the following command (from a [UI mapping](#)^[287], the [command line](#)^[316], a [macro](#)^[309], or the [clientCommand](#)^[164] method):

pin web buffer

The window identifier (**web buffer**, above) may be the title of any open external window, it may be a trailing-asterisk (*) wildcard pattern for any open external window, or it may be a single asterisk to pin all open external windows.

A pinned window displays a pin icon in the title bar in place of the Sirius icon:



You can move pinned windows around on your screen, and close or minimize them as necessary. You can also remove their pinning by:

- Selecting **UnPin** from the title bar context menu
- Issuing an [unPin](#)^[278] command

Opening the Client and external window(s) simultaneously

If you want an external window to open when the Client starts, the [Preferences](#)^[21] dialog box has options in the **Open at Startup** area to specify which Client areas (**Watch Window**, **Audit Trail**, **Web Buffer**, **Execution Trace**) you want to open as an external window when the Client starts. These options are in Client build 57 or later.

Prior to Build 57, the recommended technique for automatically opening the Client and its external windows together it was necessary to use a macro: You were to put the open command for the particular Client area (for example, `openExternalWatchWindow`) in a startup macro that you reference in the `ui.xml` file. This file, which the Client always looks for when it starts, contains any reconfigurations of the default mappings of buttons and keyboard shortcuts in the Client GUI.

You specified your startup macro as the value of the `startUpMacro` attribute in the [mappings tag in the ui.xml file](#).^[287] As described in [Creating and running a macro](#),^[309] your entire macro (the `yourMacro.macro` text file) could be as simple as the following single line:

```
openExternalWatchWindow
```

See Also

[Launching an external button bar](#)^[42]

6.6 Hiding the Client's lower windows

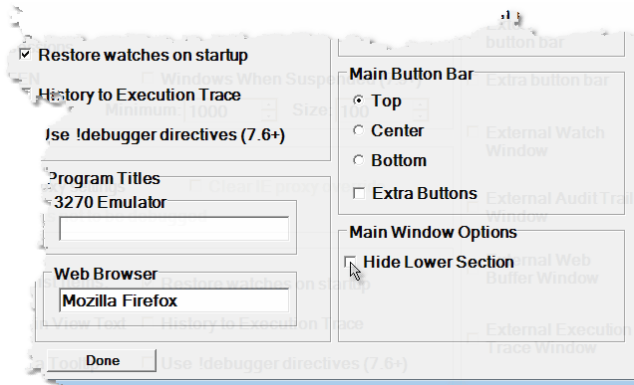
For occasions when you want more "real estate" for the main window display of the contents of one of the Client's [tabbed pages](#)^[10], you can hide the Client's [lower windows](#)^[14] and stretch the main window to occupy nearly the entire window. This is especially useful in a multiple-monitor environment, where the **Audit Trail** and **Watch Window** can reside in [separate windows](#)^[300] on another monitor.

Client Build 57 or higher is required.

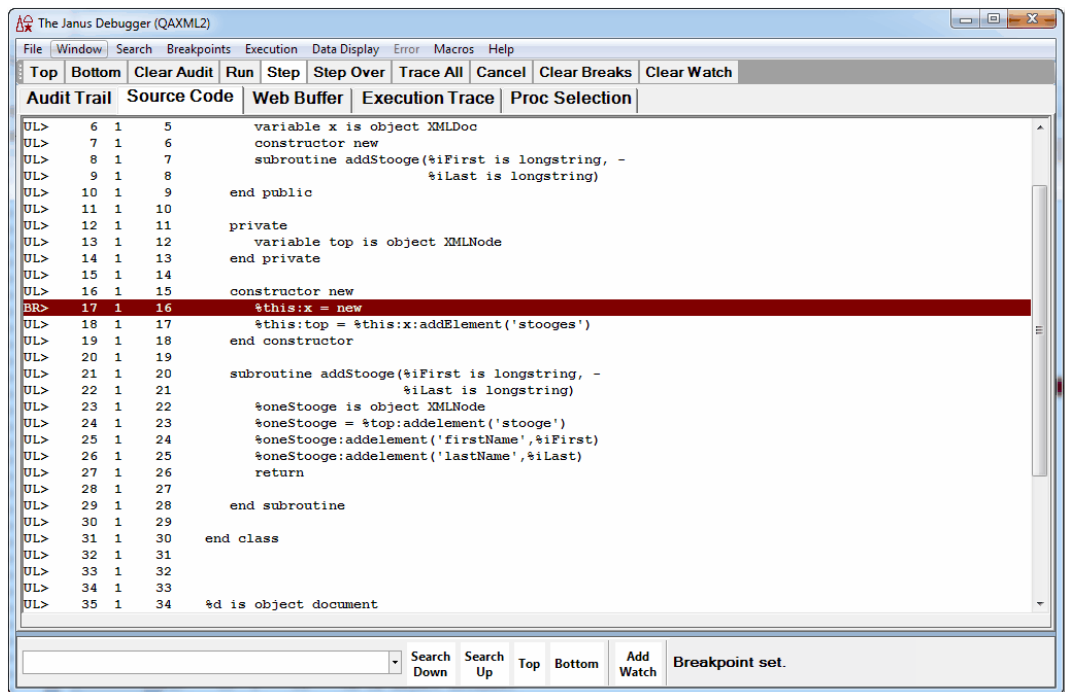
Probably the simplest way to hide the lower section of the Client is:

1. Open the **Preferences** dialog box by using the Ctrl+P keyboard shortcut (or by selecting **Preferences** from the Client's **File** menu, or by using the [preferences mappable](#)^[285] command).

2. In the **Main Window Options** section, select the **Hide Lower Section** checkbox (it is clear by default), then click **Done**.



The lower section of the Client is hidden immediately by an expanded main window:



The [searching controls](#)^[44] and the button for adding **Watch Window** items are moved to the beginning of the Status bar at the bottom of the Client. They share the single input box to the extreme left. In addition to these, you can still [right-click code lines](#)^[88] to add variables to the **Watch Window**, and you can use the searching controls available on each external window. You can also use the **Data Display** and **Search** menus, as well as the watch commands (`addWatch`, `addWatchOnCurrentLine`) and the search commands (`searchDown`, `searchFromBottom`, `searchFromTop`, `searchUp`, `top`, `bottom`) via the [Command Line tool](#)^[317].

- To restore the lower windows at any time, simply clear the **Hide Lower Section** checkbox.

You can also use Client commands to hide and restore the lower section of the main window:

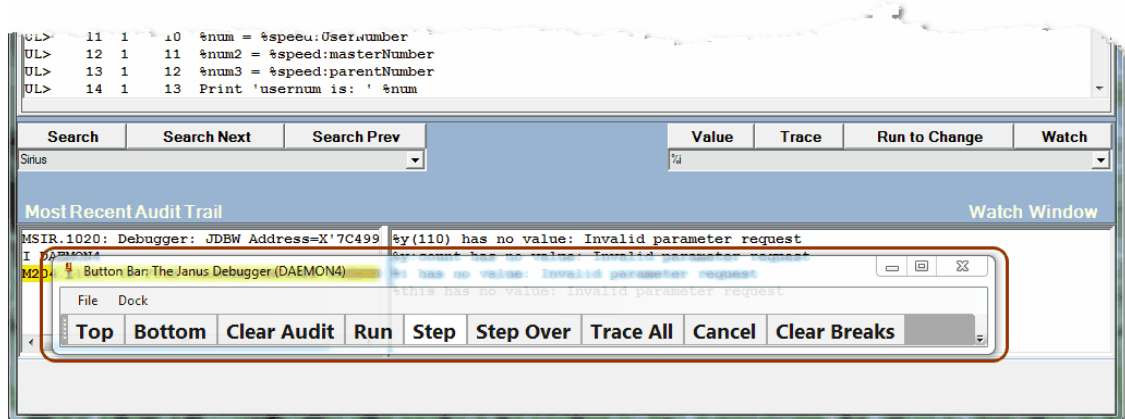
- [hideLower](#)^[213] hides the lower section if it is currently not hidden (and does nothing if it is already hidden)
- [restoreLower](#)^[243] restores a hidden lower section (and does nothing if it is not hidden)
- [toggleLower](#)^[273] either hides the lower section if it is not hidden or restores it if it is hidden

6.7 Seeing through Client windows

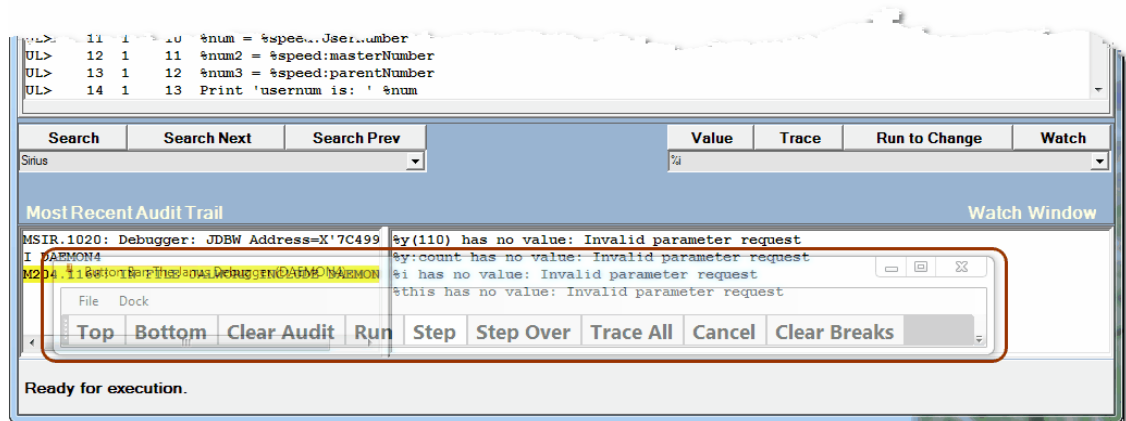
The degree of transparency of certain (Preferences and [external-button](#)^[42]) Client display windows is user-settable. You can adjust the transparency/opacity of the **Preferences** and external button windows to allow the application windows underneath to be visible.

The **opacity** element in the Client configuration file (`debuggerConfig.xml`) controls this transparency, as [described](#)^[379] in the guidelines for setting up that file.

The default opacity setting (.9) is shown below for an external button bar:



And here is a setting of .5:



The valid setting values range from .01 (least opaque) to 1 (fully opaque). Invalid values are ignored. To turn off all transparency, specify:

<opacity>1</opacity>

Client Build 59 or higher is required.

CHAPTER 7 *Using Debugger Macros*

A macro lets you execute one or more commands as a single unit of work to automate an often-repeated series of operations.

Subsequent sections in this chapter describe:

[Creating and running a macro](#)^[309]

[Mapping a macro to a button or hot key](#)^[314]

[Passing a command argument to a macro](#)^[314]

[Using the macro console and command line](#)^[316]

[Using the Macro Autorun feature](#)^[318]

[Working with macro variables](#)^[319]

[Working with Client functions](#)^[321]

7.1 **Creating and running a macro**

A Debugger macro is a Windows PC text file that contains a list of Debugger Client [commands](#)^[285] to execute (as shown in the [A macro example](#)^[312] subsection). This section describes how to set up and run a macro, as well as the context and calling requirements you must observe.

You run a macro from the [Macros menu](#)^[36], from a Client button or hot key to which you have [assigned](#)^[314] a macro, or from a [command line utility](#)^[317]. You can also run a macro automatically:

- If the macro name matches that of an included procedure.
- At Client startup, if the macro name is an attribute value at the beginning of the Client [customization file](#)^[287] (`ui.xml`) or the Client [configuration file](#)^[380] (`debuggerConfig.xml`).

Macro definition

To define a new macro:

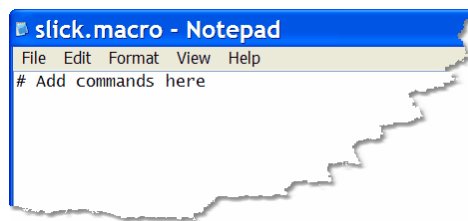
1. Start a text file by doing either of these:

- From the Client's **File** menu, select **New Blank Macro**.

The "Select name for a new macro" Windows dialog box opens, from which you select where to locate and what to name the macro file.

Macro files must have a **.macro** file extension, and the file name must not contain embedded blanks.

When you click the **Save** button, the macro file opens in *Notepad*.



- Alternatively, open a new file in a Windows text editor like *Notepad*.

When you later name and save this file after defining its contents, you must specify a **.macro** file extension and make sure the file name contains no embedded blanks.

Note: If you decide to store the file in a location other than the Client installation folder, see [Changing the location of Client work files](#).^[296]

2. In the text file, specify the Debugger commands you want to be run consecutively.

These are the formatting guidelines:

- Specify one command (case does not matter) per line; leading spaces and blank lines are ignored; line can be indefinitely long.
- Any line that begins with a number sign (#) is treated as a comment.
- Several commands require arguments; specify arguments or [argument variables](#)^[314] after the command keyword separated by at least one blank. For example:

```
addwatch %i
searchFromTop image foo
runUntil daemonnest
```

Separate multiple command arguments by one or more blanks. For example:

```
traceUntilVariableEqualsValue %i 3
```

- For a macro to run another macro, specify either of the following:

```
include macroName
```

```
macro macroName
```

Specifying the extension `.macro` after the macro name is optional. The following commands are equivalent:

```
include foo.macro
```

```
include foo
```

- Save and exit the macro file.

You can easily access this file for editing by selecting the **Edit Macro** option from the **Macros** menu.

- Consider opening [the macro console](#)^[316] (from the **Macros** menu) to display information about the macros you run.

The console reports the starting and completing of the macro execution, as well as any error messages.

- Invoke the macro.

Use any of the following ways to run a macro:

- From the **Macros** menu, select the **Run Macro** option.
- From the **Macros** menu, select the **Command Line** option.
- Use a button or hot key combination to which you have [mapped the macro](#).^[314]
- Specify the macro name as the value of the [startUpMacro](#)^[287] attribute in the Client's `ui.xml` file, or as the value of the `startup` attribute in the Client's `debuggerConfig.xml` file.

The title bar of the main Client window indicates that the macro is executing:

```
. . . macro running
```

Potential errors include: an invalid macro command, trying to invoke a macro that does not exist, and violating the [context and recursion restrictions](#).^[313]

If you want to stop a macro at any time it is running or in a not-completed state awaiting further input, select the **Kill Running Macro** option from the **Macros** menu.

Note: A macro runs until it has exhausted all its commands or encounters a [kill](#)^[218] command. If a request completes before the macro is finished, the remaining commands in the macro apply to the next request in the session. If you want to prevent these commands from being applied to the next request, use the **Kill Running Macro** option or the [noSpan](#)^[229] command.

A macro example

The following sample macro prepares a particular program (shown below the macro) for further debugging:

```
# Remove any existing breakpoints or watches
clearWatch
clearBreaks

# Set breakpoint on the first line we
# want to examine
top
breaksAt For 1 record

# watch some variables we are interested in
addWatch %i
addWatch %what

# Run to the breakpoint we set
run
```

The sample macro above was designed for a program like this:

```
begin
  %i is float
  %what is string len 30
  *BREAK
  %i = 1
  * this should not be recognized as a *break
  for 1 record
    *breaks
    %i = %i + 1
    %what = $sirTime
    change testfield to %what
  end for
  *break
  %i = %i + 1
  assert %i = 3
  trace 'hello ....'
  *break
  audit 'Hey moe...'
  print $sirtime
  print %i
end
```


Macro usage restrictions

These are the context and calling restrictions you must observe when using macros:

Context restrictions

Some of the commands that may appear in a macro are valid only when you are interactively running a program under the Debugger. If a macro attempts to execute one of these commands when no program code is ready or remains to execute, the macro is terminated, and the following message is displayed in the Client's Status bar:

Invalid context for: Offending command

The following types of commands are allowed only when executing a program:

- All run commands
- All cancel commands
- All step commands
- All watch commands
- All breakpoint commands
- All trace commands

Similarly, the following commands are allowed only after a program fails to compile. If these commands are issued when there is no compilation error, the Client issues the "Invalid context" message described above:

- nextCompileError
- previousCompileError

Recursion restriction

Macros may not be recursively called, either directly or indirectly. These sequences are not allowed:

- "Macro Alpha calls macro Alpha" (direct recursion)
- "Macro Alpha calls macro Beta which calls macro Alpha" (indirect recursion)

In either case, when a recursive call is detected, the macro is terminated and a message is displayed in the [Status bar](#)^[50].

7.2 Mapping a macro to a button or hot key

Following the same rules as for [mapping other Debugger commands](#)^[283], you can map a macro to a button or hot key. All such button/key mappings are stored [by default](#)^[296] in the `ui.xml` and `uimore.xml` files in the folder that contains the Debugger Client executable file.

To set up a button or hot key to run a macro:

1. Create an XML text file named `ui.xml` (or `uimore.xml`) as described in [Setting up the ui.xml file](#)^[287], or open the existing `ui.xml` (or `uimore.xml`) file.

2. Provide a **mapping** element that associates the macro with a button or hot key.

The value of the `command` attribute must have the following form:

```
"macro macroname"
```

where one or more blanks separate the keyword `macro` from the name of the macro.

These are examples:

```
<mapping command="macro hello" button="button2" key="m" keyModifier="ctrl"/>
<mapping command="macro    world" button="button2" buttonModifier="alt"/>
<mapping command="macro stepstep" key="Z" keyModifier="ctrl"/>
```

3. Save and close the file.

7.3 Passing a command argument to a macro

As described in [Creating and running a macro](#)^[309], several of the Debugger commands require explicit arguments when used within a macro. You can use either a standard [macro variable](#)^[319] or a standard [Client function](#)^[321] to pass an argument to a macro command at the time the macro runs.

Using the `&argstring` variable

To use the `&argstring` variable to pass an argument to a command in a macro:

1. In the macro, specify `&argstring` where you would normally specify the command argument.

For example, note the use of `&argstring` in the `breaksat` command in the following macro:

```
# Run till line that matches the user-passed string
top
clearBreaks
breaksat &argstring
run
clearBreaks
```

Note: For commands that have multiple arguments, use the numbered-argument function, `&&arg(n)`, to distinguish the arguments. For example:

```
traceUntilVariableEqualsValue &&arg(1) &&arg(2)
```

2. Provide the actual argument value before or as you run the macro.

This depends on how you invoke the macro:

- If you use the **Run Macro** option of the **Macros** menu, the contents of the [Entity-name text box](#)^[51] replace instances of `&argstring` in the commands in the macro.
- If you use the **Command Line** option of the **Macros** menu, you explicitly specify in the [command line tool](#)^[317] the replacement for `&argstring`.
- If you use an [associated](#)^[283] button or key, the `&argstring` replacement depends on whether the `macro` command in the [mapping](#)^[288] in the `ui.xml` file is specified with or without arguments:
 - If the `macro` command has an argument (after the name of the macro), that argument replaces `&argstring` in the macro. For example, if this is the mapping:


```
<mapping command="macro stooge moe" key="f2" />
```

 Pressing the F2 key invokes the `stooge` macro with `moe` as the replacement argument for instances of `&argstring`.
 - If the `macro` command is specified without an argument, the contents of the Entity-name text box replace `&argstring`.

Using the `&&prompt` function

The [&&prompt](#)^[338] function causes a macro to:

1. Pause, to accept a Client-user supplied argument value for a command that is specified within the macro
2. Continue, to execute the command with the supplied value

The format of the `&&prompt` function is:

```
&&prompt( prompt )
```

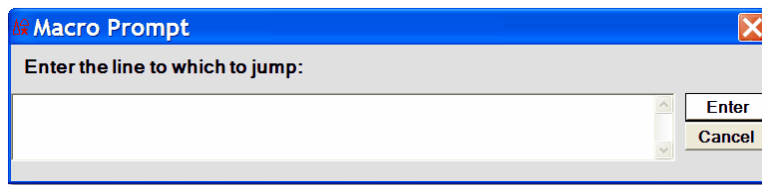
where *prompt* is either:

- A single- or double-quoted character string with 80 or fewer characters.
- A Client [macro variable](#)^[319] (requires Build 58 or higher).

As an example, the following macro clears all breaks in the current source code, prompts for the string it will use as the argument for the [breaksAt](#)^[185] command, then executes the code from its current position until it reaches a line that contains the user-supplied string, after which it clears the break:

```
# Run till first line that has the string entered at the prompt
top
clearBreaks
breaksat &&prompt("Enter the string at which to break:")
run
clearBreaks
```

When the macro command that contains `&&prompt` executes, the Client displays a **Macro prompt** dialog box like the following, which shows the prompt string from the preceding example macro:



You can use the `&&prompt` function wherever an argument to a command may appear, as shown in the following example:

```
traceUntilVariableEqualsValue &&prompt('var') &&prompt("val")
```

When this command executes, it produces two consecutive prompts, one for each of the command arguments.

7.4 Using the console and command line

As macro development aids, the Debugger Client provides a console window and a command line dialog box.

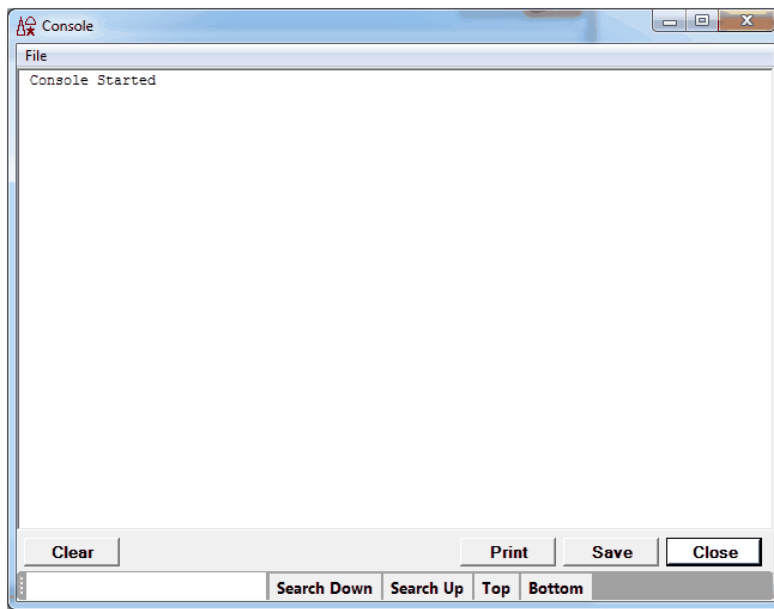
The Console

The **Console** is an independent window that logs informational, error, and trace messages from the Debugger macros (and Client commands) that you run.

To use the console:

1. From the Client's **Macros** menu, select **Console**.

The **Console** window opens. It remains open until you explicitly close it.



2. [Invoke a macro](#)^[309] or [invoke command](#)^[179].

The console window moves to the top of the window stack and displays information about the macro or command, including its starting and stopping, as well as any error messages.

The console's **Print** and **Save** buttons and **File** menu options work the same as their counterparts in the Client [Text Viewer](#)^[149]. The **Close** button closes the window. The search bar on the bottom of the window provides controls for searching the window content.

You can also clear the window with the [clearMacroConsole](#)^[191] command, and you can close it with the [closeMacroConsole](#)^[196] command.

Note: If you have the console open, [value displays](#)^[101] appear in the console instead of in a **Value** window. To override this default, use the `valueDisplayOnConsole` option of the Client [setPreference](#)^[260] command.

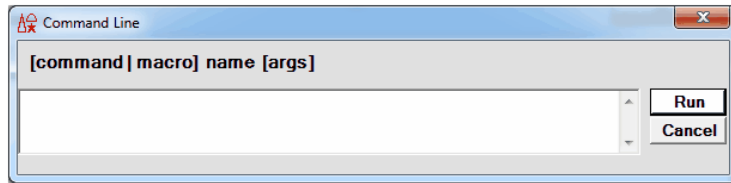
The Command Line

The **Command Line** dialog box provides a command line interface for running macros and commands. This utility is an alternative to the **Macros** menu **Run Macro** option or to a button or hot key combination; it stays available unless explicitly closed, so it is a step-saver for macro testing. In addition, this interface will invoke a [Client command](#)^[179] if no same-named macro exists.

To use the command line tool:

1. Consider opening [the Console window](#)^[316] (from the Client's **Macros** menu) to display information about the macros or commands you run.
2. From the **Macros** menu, select **Command Line**.

The **Command Line** dialog box opens. It remains open until you explicitly close it.



3. Enter the name of a macro or command, and specify its required arguments, if any.

The extension **.macro** after a macro name is optional, as is a preceding **Macro** keyword (case not important). The **Command** keyword directs the Client to look exclusively for a command that has the name you are specifying.

For a macro, the Client looks for the macro file in the same folder as the **JanusDebugger.exe** file, [by default](#)^[296].

Use one or more blanks to separate arguments from each other and from the macro or command name. If you provide more arguments than are required, the extra characters are ignored.

4. Click the **Run** button to execute the named macro or command.

Clicking **Cancel** closes the command line dialog box.

5. If during your session you run many macros and commands, you can easily review them or repeat an earlier one by scrolling through their history using your keyboard up/down arrow keys (Client Build 59 or higher required).

7.5 Using the Macro Autorun feature

The Debugger Client lets you automatically run a particular macro whenever you debug a particular procedure. Such a macro might set up watches and breakpoints tailored to aid the debugging of that procedure. This Macro Autorun feature requires simply that the macro have the same name as the procedure, and it works only for procedures that are included from command level (level 0), that is, **not** from within a procedure (level 1).

By default, the feature is initially not enabled.

To use the feature:

1. For a procedure that you debug somewhat frequently, [define](#)^[309] a macro that sets up the debugging environment you want for that procedure.

2. Give the macro the same name as the procedure. Character case is not important.
3. From the keyboard or from the Client **File** menu, select the **Preferences** option.
4. In the **Execution Options** section of the **Preferences** dialog box, select the **Macro Autorun** checkbox, which is clear by default.

Note: The Client [setPreference](#)^[260] command has an option that lets you toggle the **Macro Autorun** checkbox.

5. Include the procedure, for example via the URL of your Janus Web application.

The Debugger searches (the Debugger installation folder or in any [alternative work folder](#)^[296] you may have specified) for a **.macro** file that has the same name as the procedure. If such a file is found, it is run.

If a same-named macro is not found, no action is taken.

7.6 Working with macro variables

Macro variables are placeholders for Debugger command arguments within Debugger macros, as well as placeholders for Client function arguments (as of Client Build 58). Distinguished by names that begin with a single ampersand (&) character, macro variables can be set to a variety of types of values.

As an example, you could have a macro set some breakpoints, run to each of them, and use a macro variable to note the value of the program's %variable at each break. This would let you note the values at different points in program execution:

```
breaks
run
set &val1 = %i
run
set &val2 = %i
echo &val1
echo &val2
```

[set](#)^[255] (in the example above) is a Client command that is particularly useful for working with macro variables.

Macro variables are either system-supplied (predefined) or user-defined.

Predefined macro variables

The system **&argstring** variable is described in [Passing a command argument to a macro](#).^[314]

User-defined macro variables

To define your own macro variable, you use the [set](#)^[255] command within a macro. As shown in the syntax and examples below, the types of values to which you can set the variable include constants, User Language variables, fields, and \$list elements, and other macro variables and functions.

The syntax of the `set` command is:

```
SET &target = "string" | [-]nnn | &var | %xxx | g.xxx | f.xxx  
| $listcnt(x) | $listinf(x,y) | &&function
```

where:

- `&target` is the case-sensitive name of a macro variable to create or set. If no variable with that name exists, it is created and set. If a variable with that name already exists, its value is reset.

The name must begin with a single ampersand (&), which must be followed by an alphabetic character, which is optionally followed by one or more alphanumeric characters and underscores. For example, `&A_b` is a valid name, but `&` and `&a.b` are *not* valid.
- `"string"` is a string constant.
- `[-]nnn` is an integer constant with an optional leading minus sign.
- `&var` is a previously defined macro variable.
- `%xxx` is a User Language percent variable.
- `g.xxx` is a Debugger [global variable reference](#)^[96].
- `f.xxx` is a Debugger [field reference](#)^[95], possibly with a subscript.
- `$listcnt/$listinf` are the Debugger functions for [viewing \\$list counts and elements](#).^[98]
- `&&function` is a macro function.

These command examples also show that multiple blanks may surround names and values:

```
SET &a2 = %a(2)  
  
set &i=%i  
  
SEt   &g =   %G  
  
set &s=%s  
  
set &l =   %L  
  
set &lc = $listcnt(%g)  
  
set &l1 = $listinf(%g,1)
```



```

set &l2 = $listinf(%g,2)

set &global = g.JACK

set &sl = %sl:item(1)

set &num    = 1

set &string = "a"

set &a = &&prompt("g'day mate!")

set &f  = f.name

set &f3 = f.name(3)

set &a2 = -1

set &xx = &asa

```

Note: If you later want to review the values of macro variables you have defined, you can issue the [varDump](#)^[280] Client command, for example from the [Command Line](#)^[317] tool. Other commands useful for working with macro variables include [setM204Data](#)^[260], [assert](#)^[180], [continueIf](#)^[197], and [toggle](#)^[271].

7.7 Working with Client functions

The available Debugger Client functions are described in individual subsections that follow. The function names are specified without regard for case.

Client function names begin with two ampersand (&&) characters; those characters must be followed by one alphabetic character, which may be followed by one or more alphanumeric characters and underscores.

Client function arguments *all* have the same form: that is, they may be single- or double-quoted strings, numeric constants, or, as of Client Build 58, they may also be [macro variables](#)^[319].

To see in a message box or the console a value returned by a function, you can assign it to a macro variable, then issue the [varDump](#)^[280] command; or use the function with the [echo](#)^[203] command.

Macro-only functions

It may be that a &&function is **macro-only**: it may be used only in a Debugger [macro](#)^[309]. The description of such a &&function includes a **Scope** section that reminds of this restriction.

String functions

The `&&`functions include a group of string manipulation functions, all of which follow these rules:

- The first character in a string occupies position 1, the second occupies 2, and so on.
- If a function searches for and returns the position of a desired string within a target, it returns 0 if the searched-for string is not found.
- If a function takes a position or length as an argument, and the argument value that is passed is non-numeric, an error is issued and the command that references the function is aborted.
- If a number is passed for a parameter that is a string, the number is converted to a string. For example, 1234 is treated like '1234'.
- Character matching is case sensitive.
- Like all `&&`functions, arguments may be single- or double-quoted strings, numeric constants, or `&`variables.

7.7.1 `&&amDaemon` function

Action Useful for testing purposes, this function returns a 1 if a Daemon tab contains the currently active code. Otherwise, the returned number is 0.

Syntax:

`&&amDaemon`

Introduced Build 59

7.7.2 &&arg function

Action Serves as a placeholder for a command argument *within a macro* (if used outside of a macro, an error is issued). The argument that takes the place of **&&arg** is [dynamically provided](#)^[315] by the Client user.

Syntax:

&&arg(*n*)

where *n* is a single- or double-quoted string, a numeric constant, or as of Build 58, a [macro variable](#)^[319].

Designed for commands that have multiple arguments, this function parses the blank-delimited, user-provided string to determine the replacement values for the **&&arg** occurrences within the command. The first such value in the string replaces **&&arg(1)**, the second replaces **&&arg(2)**, and so on.

For example, for the following provided argument string, **&&arg(1)** returns **%a**:

%a 2

The [&argstring](#)^[314] variable is a placeholder designed for single-argument commands within a macro.

If a command takes a single argument, or you want to treat whatever is passed to the command as a single string (even if it contains blanks), use the [&argstring](#)^[314] variable instead of the **&&arg** function.

Scope Allowed only in Debugger macros

Introduced Build 28

7.7.3 **&&assertFailureCount** function

Action Returns a count of the number of times that the result of an [assert](#)^[180] command is a Failure (since the beginning of the execution of the macro that contains the **assert**).

Syntax:

&&assertFailureCount

The "assert" functions (also including **&&assertSuccessCount** and **&&assertStatus**) are particularly useful if you are using the **assert** command to automate your code testing, letting you keep track of the successes and failures of the assertions.

To clear the count at any time (other than by default when the Client is started/restarted or when a new macro is invoked), you can use the [resetAssertCounts](#)^[241] command, as of Build 56.

Introduced Build 50

7.7.4 **&&assertStatus** function

Action Returns a string that contains a summary of the counts of [assert](#)^[180] command results (since the beginning of the execution of the macro that contains the **assert**).

Syntax:

&&assertStatus

The "assert" functions (also including **&&assertSuccessCount** and **&&assertFailureCount**) are particularly useful if you are using the **assert** command to automate your code testing, letting you keep track of the successes and failures of the assertions.

To report the function result, you may want to use the [echo](#)^[203] command, as shown in the following [macro trace output](#)^[222]:

```
>>>macroTrace: macroTrace on
>>>macroTrace: echo &&assertStatus
Macro message: Assert Summary: Failed: 0, Succeeded: 17
```

Introduced Build 50

7.7.5 **&&assertSuccessCount** function

Action Returns a count of the number of times that the result of an [assert](#)^[180] macro command is a Success (since the beginning of the execution of the macro that contains the **assert**).

Syntax:

&&assertSuccessCount

The "assert" functions (also including **&&assertFailureCount** and **&&assertStatus**) are particularly useful if you are using the **assert** command to automate your code testing, letting you keep track of the successes and failures of the assertions.

To clear the count at any time (other than by default when the Client is started/restarted or when a new macro is invoked), you can use the [resetAssertCounts](#)^[241] command, as of Build 56.

Introduced Build 50

7.7.6 **&&blackOrWhiteList** function

Action

Tests whether the Debugger is [filtering](#)^[81] the procedures you are going to debug, and if so, whether a Black List or White List is being used.

Syntax:

&&blackOrWhiteList

The all-lowercase return string is one of these values: **black**, **white**, or **none**.

Example:

The following macro includes a comparison involving **&&blackOrWhiteList**:

```
continueMacroIf &&blackOrWhiteList <> 'black'  
labelButton button0 BlackList Off  
turnOnBlackList  
setTitle Black List on  
set &changed = 1  
clearStatus
```

Introduced

Build 62

7.7.7 &&concatenate function

Action Concatenate and return the function's arguments.

If fewer than two arguments are specified, an error is issued.

Syntax:

```
&&concatenate( str1, str2[, str3] ...)
```

where the *str* arguments observe the [rules for &&function string arguments](#)^[322].

For example, the following fragment would display: **abc123hi**

```
set &hello = 'hi'  
set &result = &&concatenate('a', 'b', "c", 1,2,3,&hello)  
echo &result
```

Introduced Build 58

7.7.8 &&currentRunningMacro function

```
<topicref type="topic" id="411978089953851" build="ALL"
modified="2012-10-15T15:15:03.856Z" icon="0"
href="currentRunningMacro_function">
```

```
  <caption translate="true">&&currentRunningMacro
function</caption>
```

```
  <topic><![CDATA[Ã-Â»Â¿<?xml version="1.0"
encoding="UTF-8"?>
```

```
<?xml-stylesheet type="text/xsl" href=" ../helpproject.xsl" ?>
```

```
<topic template="Default" lasteditedby="JAL"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" ../helpproject.xsd">
```

```
  <title translate="true">&&currentRunningMacro
function</title>
```

```
  <keywords>
```

```
    <keyword
```

```
translate="true">&&currentRunningMacro function,
Debugger Client</keyword>
```

```
  </keywords>
```

```
  <body>
```

```
    <header>
```

```
      <para styleclass="Heading1"><text styleclass="Heading1"
translate="true">&&currentRunningMacro
function</text></para>
```

```
    </header>
```

```
      <para styleclass="Para"><table styleclass="cmdref"
rowcount="2" colcount="2" style="width:100%;
page-break-inside:avoid;">
```

```
        <tr style="vertical-align:top">
```

```
          <td style="width:20%;">
```

```
            <para styleclass="Para"><text style="font-family:Verdana
font-size:11pt; color:#660000;"
translate="true">Action</text></para>
```

```
          </td>
```

```
        </tr>
```

```
      <para styleclass="Para"><text styleclass="Para"
translate="true">Returns the full file-system path to the macro
within which it is called. If not called from within a macro, the
function returns a null string. </text></para>
```

```
      <para styleclass="Para" style="page-break-inside:avoid;
page-break-after:avoid;"><text styleclass="Para"
translate="true">Syntax: </text></para>
```

```
      <para styleclass="Code Example"><text styleclass="Code
Example"
translate="true">&&currentRunningMacro</text></para>
```

7.7.9 &¤tTitle

Action Returns the current title of the Client main window.

Syntax:

&¤tTitle

This function may be useful in conjunction with the [setTitle](#)^[264] and [restoreTitle](#)^[243] commands.

See also [&&originalTitle](#)^[337].

Introduced Build 62

7.7.10 **&&exists** function

Action Tests if a [macro variable](#)^[319] is defined.

Syntax:

```
&&exists(variableName)
```

where *variableName* is the macro variable whose existence is being checked.

If the named variable exists, **&&exists** returns **1**; otherwise, it returns **0**.

A variable is deemed to "not exist" in these cases:

- It has not been defined since the Debugger Client was last started or restarted.
- It has been defined, but it was removed with the **unset** command.

Examples:

The following statement returns **0**:

```
echo &&exists("&neverSet")
```

The **echo** statement below returns **1**:

```
set &i = 666  
echo &&exists("&i")
```

The **echo** statement below returns **0**:

```
unset &i  
echo &&exists("&i")
```

Introduced Build 60

7.7.11 **&&getMainSearchInputArea** function

Action Returns the value specified in the [search](#)^[44] input area on the Client main window. If the Client is in [hide-lower](#)^[305] mode, the function returns the value specified in the input box in the [Status bar](#).^[50]

Note: The function does **not** return a value for an external window.

Syntax:

&&getMainSearchInputArea

Introduced Build 58

7.7.12 **&&getVariableOrFieldInputArea** function

Action Returns the value specified in the [Entity-name input box](#)^[51]. If the Client is in [hide-lower](#)^[305] mode, the function returns the value specified in the input box in the [Status bar](#).^[50]

Syntax:

&&getVariableOrFieldInputArea

Introduced Build 58

7.7.13 **&&globalAssertFailureCount** function

Action Same as [&&assertFailureCount](#)^[324] except its scope is the entire Client session. During such a session, the count is only cleared if done explicitly with the [resetGlobalAssertCounts](#)^[241] command.

Syntax:

&&globalAssertFailureCount

Introduced Build 57

7.7.14 **&&globalAssertStatus** function

Action Same as [&&assertStatus](#)^[324] except its scope is the entire Client session. During such a session, the counts are only cleared if done explicitly with the [resetGlobalAssertCounts](#)^[241] command.

Syntax:

&&globalAssertStatus

Introduced Build 57

7.7.15 **&&globalAssertSuccessCount** function

Action Same as [&&assertSuccessCount](#)^[325] except its scope is the entire Client session. During such a session, the count is only cleared if done explicitly with the [resetGlobalAssertCounts](#)^[241] command.

Syntax:

&&globalAssertSuccessCount

Introduced Build 57

7.7.16 **&&ieMode** function

Action Returns the current setting of the Client's [IE Mode preference](#).^[19]

Syntax:

&&ieMode

The returned value is a lowercase string: **none**, **proxy**, **newpac** or **mergedpac**. These values are exactly the options of the [setIEmode](#)^[258] command.

Introduced Build 62

7.7.17 **&&index** function

Action Returns the 1-based position of the *needle* argument within the *haystack* argument, or it returns **0** if the *needle* value is not found within the *haystack* value.

If fewer than two arguments are specified, an error is issued.

Syntax:

&&index(*haystack*, *needle*)

where the *haystack* and *needle* arguments observe the [rules for &&function string arguments](#)^[322].

For example, the first **&&index** call in the following fragment would return **3**; the second would return **0**:

```
set &g = 'george'
&&index(&g, 'or')
&&index('moe', 'x')
```

Introduced Build 58

7.7.18 **&&isWatched** function

Action Determines whether its argument is a variable that is currently specified in the **Watch Window**. If it is, **&&isWatched** returns **1**; if not, it returns **0**.

Syntax:

&&isWatched(*value*)

where *value* is a single- or double-quoted string, a numeric constant, or a [macro variable](#)^[319].

Introduced Build 58

7.7.19 **&&length** function

Action Returns the length in characters of its string argument.

Syntax:

```
&&length( value )
```

where *value* observes the [rules for &&function string arguments](#)^[322].

For example, the first **&&length** call in the following fragment returns 0; the second returns 3; the last returns 6:

```
&&length("")  
&&length(123)  
set &g = 'george'  
&&length(&g)
```

Introduced Build 58

7.7.20 **&&numberOfBreakpoints** function

Action Returns the number of breakpoints that are set in the current request, irrespective of the current executing position in the request.

Syntax:

```
&&numberOfBreakpoints
```

Introduced Build 58

7.7.21 **&numberOfLevels** function

Action Returns the number of code levels being debugged. The main program is one level and each active daemon adds another level. For example, if a request spawns a daemon which itself spawns a daemon, and that second daemon still has code to execute, the return to **&numberOfLevels** is 3.

If no program is being debugged, 0 is returned.

Syntax:

&numberOfLevels

Introduced Build 59

7.7.22 **&numberWatched** function

Action Returns the number of items that are currently [being watched](#)^[87] in the **Watch Window**.

Syntax:

&numberWatched

Introduced Build 58

7.7.23 **&&originalTitle**

Action Returns the default value of the title of the Client main window (the window title is subject to change by a [setTitle](#)^[264] command). The default main window title is "The Janus Debugger" or "The Sirius Debugger."

Syntax:

&&originalTitle

The **&&originalTitle** function returns the same value as the [restoreTitle](#)^[243] command.

See also [&¤tTitle](#)^[330].

Introduced Build 62

7.7.24 **&&preference function**

Action Returns the value of the specified Client preference setting. Client preferences are selected from the [Preferences](#)^[18] dialog box or the [Proc Selection](#)^[13] page, or they are set with the [setPreference](#)^[260] command. Their names for the purposes of this function match the options listed for the **setPreference** command.

Syntax:

&&preference(*preferenceName*)

where *preferenceName* is one of the **setPreference** command preference options (not case-sensitive).

For example, the following command returns a **1** or **0** (indicating that the preference is on or off):

echo &&preference('BreakAfterReadScreen')

Introduced Build 60

7.7.25 **&&procName** function

Action Returns the name of the procedure that is being executed. If no procedure or an ad hoc procedure is being executed, a zero-length string is returned.

Syntax:

&&procName

Introduced Build 48

7.7.26 **&&prompt** function

Action Pauses a macro while you supply an argument value for a command that is specified within the macro, then continues running the macro using the supplied value for the command.

Syntax:

&&prompt(" *string*" | &*var*)

where:

- *string* is any single- or double-quoted character string.
- &*var* is a [macro variable](#)^[319]. This option is new in Build 58.

Described further in [Passing a command argument to a macro.](#)^[315]

Introduced Build 27

7.7.27 **&&searchResult** function

Action Returns the line number of the line that contained the found string, if the last Client search operation (invoked by **Search** button or command) successfully found something. If the last search found nothing, or if there was no prior search, then **&&searchResult** returns **-1**.

Syntax:

&&searchResult

This function considers the numbering of the lines in a Client tab's display to start with 0.

Introduced Build 48

7.7.28 **&&searchSuccess** function

Action Returns **1** (True) if the last Client search operation (invoked by button or command) successfully found something. If the last search found nothing, or if there was no prior search, **&&searchSuccess** returns **0** (False).

Syntax:

&&searchSuccess

Introduced Build 48

7.7.29 **&&selectedTab** function

Action Useful for testing purposes, this function returns the label of the Client main window tab that is currently active (for example, "Audit Trail"). Or, it returns a null string ("") if no tab is active (for example, no program is being debugged).

Syntax:

&&selectedTab

Introduced Build 59

7.7.30 **&&statusMessage**

Action Returns the most recent Client [status](#)^[50] message.

Syntax:

&&statusMessage(*windowName*)

where *windowName* is a single- or double-quoted string, a numeric constant, or a [macro variable](#)^[319] that identifies a Client window. *windowName* may be specified without regard for character case.

Introduced Build 59

7.7.31 **&&substring** function

Action Return a substring of the target *string* argument. The returned substring characters begin with the *start* character position (1-based) and continue for *len* characters. If no *len* value is specified, the returned substring extends to the end of *string*.

Syntax:

```
&&substring( string, start[, len])
```

where the arguments observe the [rules for &&function string arguments](#)^[322].

For example, the first **&&substring** call in the following fragment returns **b**; the second returns **bc**; the last returns **c**:

```
&&substring('abc',2,1)
&&substring('abc',2)
&&substring('abc',3)
```

Introduced Build 58

7.7.32 **&&sum** function

Action Returns the sum of the function arguments.

Syntax:

```
&&sum( value, value ...)
```

where *value* is a numeric constant, or a [macro variable](#)^[319].

If fewer than two arguments are passed, or if any argument has a non-numeric value, an error is issued.

See also the [increment](#)^[215] and [decrement](#)^[201] commands.

Introduced Build 58

7.7.33 **&&verifyMatch** function

Action Returns the 1-based position (in *string*) of the first character in *string* that is also in the characters in *charSet*. If no character in *string* is in *charSet*, returns 0.

Syntax:

&&verifyMatch(*string*, *charSet*)

where the arguments observe the [rules for &&function string arguments](#)^[322].

For example, the first **&&verifyMatch** call in the following sequence returns 2; the second returns 1; the third returns 1; the fourth returns 3; the fifth returns 0:

```
&&verifyMatch("Shazam", "abcsh")
&&verifyMatch("Shazam", "abcSh")
&&verifyMatch("Shazam", "Shazam")
&&verifyMatch("Shazam", "abc")
&&verifyMatch("Shazam", "xxx")
```

Introduced Build 58

7.7.34 **&&verifyNoMatch** function

Action Returns the 1-based position (in *string*) of the first character in *string* that is **not** in the characters in *charSet*. If no character in *string* is not in *charSet*, returns 0.

Syntax:

```
&&verifyNoMatch( string, charSet )
```

where the arguments observe the [rules for &&function string arguments](#)^[322].

For example, the first **&&verifyNoMatch** call in the following sequence returns 1; the second returns 4; the third returns 0; the fourth returns 4 :

```
&&verifyNoMatch("Shazam", "abcsh")
&&verifyNoMatch("Shazam", "abcSh")
&&verifyNoMatch("Shazam", "Shazam")
&&verifyNoMatch("123.45", "0123456789")
```

Introduced Build 58

7.7.35 **&&windowStatus** function

Action Returns 1 if the named window is [open](#)^[300], 0 otherwise.

Syntax:

```
&&windowStatus( windowName )
```

where *windowName* is a single- or double-quoted string, a numeric constant, or a [macro variable](#)^[319] that identifies a Client window. *windowName* may be specified without regard for character case.

Introduced Build 58

CHAPTER 8 *Problem Diagnosis*

These sections are included:

[Debugging the Janus Debugger](#)^[345]

[Debugging the Sirius Debugger](#)^[348]

[How the Sirius Debugger handles communication breaks](#)^[357]

[Tracking Client performance](#)^[359]

[Resolving issues when automatically maintaining IE proxy settings](#)^[361]

8.1 Debugging the Janus Debugger

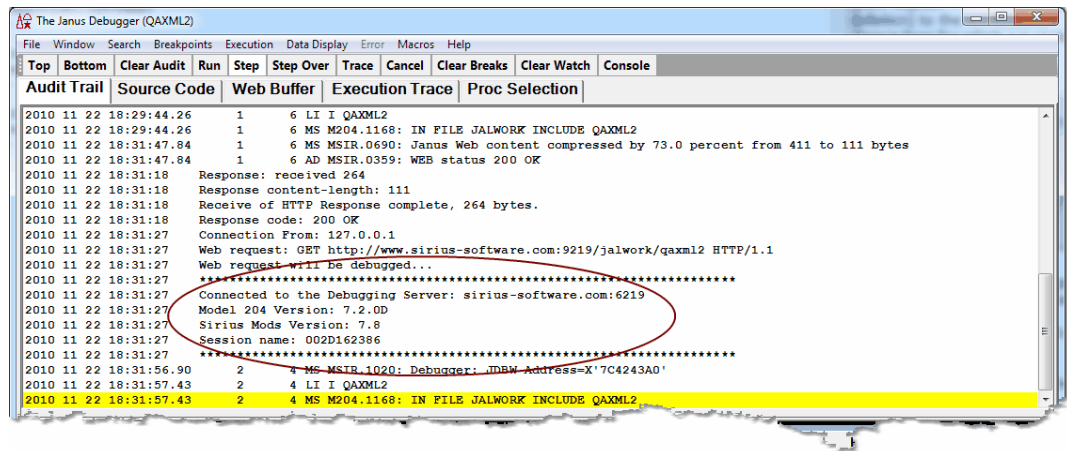
Whenever your browser sends a web request to a *Janus Web Server* that has been configured for debugging, this sequence of events unfolds:

1. Before sending your web request, the Debugger Client on the workstation, which is [defined](#)^[380] as your proxy server, connects to the Debugger Server port on the online (set up during product [installation](#)^[367]). The Debugger Server starts a *Model 204* thread whose default user ID is SOCKUSER.
2. The Debugger Server communicates over a socket connection with the Client and creates a worker thread for this debugging session. The connection handshake contains a unique ID from the Client for this debugging session. The Server worker thread uses AUDIT statements to report its activities to the *Model 204* audit trail.

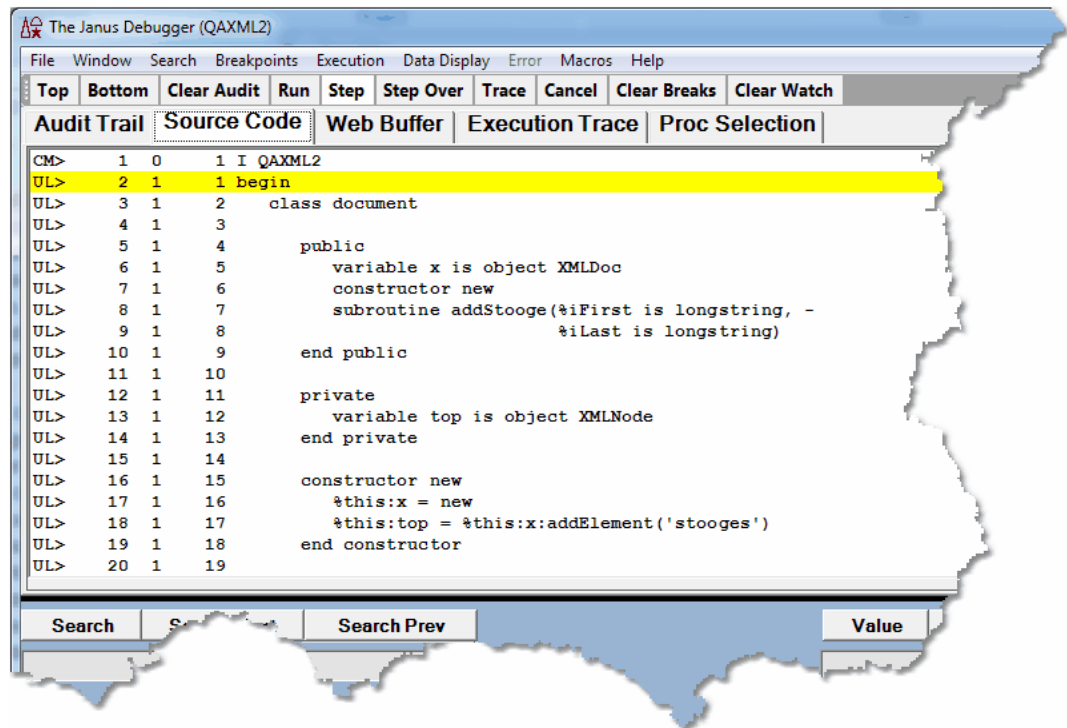
The following *SirScan* audit trail lines for user SOCKUSER report the Debugger Server activity described thus far. The session ID is encircled. Subsequent worker thread lines are shown [later](#)^[347]:

```
3 AD MSIR.0114: Processing connection for port DEBUGSERVER3355 from 198.242.244.16
3 AD M204.0352: IODEV=15, OK SOCKUSER SOCKUSER LOGIN 05.363 DEC 29 10.44.30 -13
3 LI I DEBUGSERVER.UL
3 MS M204.1168: IN FILE JALPROC INCLUDE DEBUGSERVER.UL
3 MS M204.1168: IN FILE JALPROC INCLUDE DEBUGSERVER.UTABLE
3 MS M204.1168: IN FILE JALPROC INCLUDE DEBUGSEVERTOOLS.UL
3 ST USERID='SOCKUSER' ACCOUNT='SOCKUSER' LAST='CMPL' SUBSYSTEM='' PROC-FILE='JALPROC'
    NTBL=97 QTBL=584 STBL=5690 UTBL=776 PDL=696 CPU=15 DKRD=4 PCPU=832 RQTM=18 DKPR=12
3 US Debug request from: 198.242.244.16 33D39348
```

- The worker thread is directed by the Debugger Client (based on user interface actions), and the worker thread maintains an internal master/slave, post/wait relationship with the Web Server thread being debugged. The initial action of the worker thread is to announce itself to the Debugger Client in the **Audit Trail** page:



- The worker then continues in a loop/dialogue with the Client, reporting its state and latest activity to the Client, and receiving XML requests from the Client (commands that are based on what the Debugger GUI user is invoking). If the normal Web Server response to your web request is to include a *Model 204* procedure, for example, the Client sends the worker a command to have the Web Server continue, the worker posts that command to the Web Server, the worker sends AUDIT lines about this exchange, and the worker waits.
- While the worker thread waits, the Debugger Client sends your web request to the Janus Web Server. The worker's wait limit is two minutes, so it will time out and end if the request fails for any reason. Otherwise, the Web Server recognizes the debugging session ID, so sends its response to the worker. In this example, the response is procedure code, which is sent to the worker instead of being run. The worker wakes up, sends the procedure code in an XML document to the Client, and the Client displays the procedure code in the Debugger **Source Code** page:



The worker thread reports its actions thus far in the lines below that begin with three asterisks (***). The worker refers to the Web Server thread as the "debuggee":

```

3 AD MSIR.0114: Processing connection for port DEBUGSERVER3355 from 198.242.244.16
3 AD M204.0352: IODEV=15, OK SOCKUSER SOCKUSER LOGIN 05.363 DEC 29 10.44.30 -13
3 LI I DEBUGSERVER.UL
3 MS M204.1168: IN FILE JALPROC INCLUDE DEBUGSERVER.UL
3 MS M204.1168: IN FILE JALPROC INCLUDE DEBUGSERVER.UTABLE
3 MS M204.1168: IN FILE JALPROC INCLUDE DEBUGSERVERTOOLS.UL
3 ST USERID='SOCKUSER' ACCOUNT='SOCKUSER' LAST='CMPL' SUBSYSTEM='' PROC-FILE='JALPROC'
   NTBL=97 QTBL=584 STBL=5690 UTBL=776 PDL=696 CPU=15 DKRD=4 PCPU=832 RQTM=18 DKPR=12
3 US Debug request from: 198.242.244.16 33D39348
3 US *** Connection to worker: External user number: 5 Internal user number: 2
3 US *** Worker got command from client: R
3 US *** Worker posts debuggee, and then waits...
3 US *** Debugger Worker wakes up: S
    
```

6. The Client-worker-Web Server communication continues in this fashion according to the commands invoked by the Client GUI user. The Client user initiates the next round of activity by invoking an operation on the source code, say, stepping to the next statement. The Client signals the worker, and the worker instructs the Web Server thread. The worker reports a sequence of audit lines similar to the previous ones (the ending **N**'s below mean "run to **N**ext statement" and "**N**ext statement executed"):

```
RK SCAN: US=SOCKUSER TM=C6F2F410 IP=198.242.244.16 JA=DEBUGSERVER3355
US ***Worker got command from client: N
US ***Worker posts debuggee, and then waits...
US ***Debugger Worker wakes up: N
```

7. In case you need to debug the Debugger, you can access the XML traffic exchanged between the worker and Debugger Client. To do so, use the JANUS TRACE command (described in the *Janus TCP/IP Base Reference Manual*) to increase the tracing on the Debugger Server port (in this example, DEBUGSERVER3355), and use *SirScan* to view the traffic. The JANUS DISPLAYTRACE command reveals the current trace value.

Since a high tracing value, say **15**, can capture huge amounts of data, remember to return the tracing setting to its former value when you no longer need so much detail.

Also of possible use in a debugging situation, the Debugger Client installation folder is the [default location](#)^[296] for a text log (**log.txt**) that includes the Client and workstation browser activity that is captured by the Client.

8.2 Debugging the Sirius Debugger

Whenever you invoke a User Language program from an Online that has been configured for debugging, this sequence of events unfolds:

1. The initial user call to start the Sirius Debugger (**SIRDEBUG ON DEBCLI1 198.242.244.16 8081 3270** in the *SirScan* example lines below, which is [described](#)^[388] as part of the product installation) triggers a socket connection request from the issuing user's thread (using the CLSOCK port [defined](#)^[368] during installation) to the workstation that hosts the Debugger Client. The connection request (trace lines for which are circled below) contains the number of the Debugger Server port (set up during product [installation](#)^[367]) and the user thread number.

```
18 LI SIRDEBUG ON DEBCLI1 198.242.244.16 8081 3270
18 MS MSIR.0469: Establishing connection to host 198.242.244.16 port number 8081 using
   Janus CLSOCK port DEBCLI1, user socket number 2
18 RK APO(2) 000000 : 47455420 2F204854 54502F31 2E310D0A | GET / HTTP/1.1..
18 RK APO(2) 000010 : 69676E6F 72654C65 76656C5A 65726F3A | ignoreLevelZero:
18 RK APO(2) 000020 : 20310D0A 4A616E75 734F6E6C 696E6552 | 1..JanusOnlineR
18 RK APO(2) 000030 : 65717565 73743A20 30303031 380D0A57 | equest: 00018..W
18 RK APO(2) 000040 : 6F726B65 72506F72 743A2030 33323730 | orkerPort: 03270
18 RK APO(2) 000050 : 0D0A0D0A | .....
```

- Once the connection succeeds, the Debugger Client, in turn, connects to that Debugger Server port, which starts a *Model 204* thread whose default user ID is SOCKUSER (see top two *SirScan* lines, below). The Debugger Server communicates over a socket connection with the Client and creates a worker thread for this debugging session. The connection handshake contains a unique ID from the Client for this debugging session (circled, below). The Server worker thread uses AUDIT statements to report its activities to the *Model 204* audit trail.

```

2 AD MSIR.0114: Processing connection for port DEBUGSERVER3270 from 198.242.244.16
2 AD M204.0352: IODEV=15, OK SOCKUSER SOCKUSER LOGIN 06.170 JUN 19 14.29.14 1 C6F2F410
2 LI I DEBUGSERVER.UL
2 MS M204.1168: IN FILE JALPROC INCLUDE DEBUGSERVER.UL
2 MS M204.1168: IN FILE JALPROC INCLUDE DEBUGSERVER.UTABLE
2 MS M204.1168: IN FILE JALPROC INCLUDE DEBUGSEVERTOOLS.UL
2 ST USERID='SOCKUSER' ACCOUNT='SOCKUSER' LAST='CMPL' SUBSYSTEM='' PROC-FILE='JALPROC' PROC='DEBUGSERVER.UL'
   NTBL=100 QTBL=606 STBL=5712 UTBL=785 PDL=700 CPU=14 PCPU=932 RQTM=15 DKPR=14
2 RK APO 000000 : 32303020 4A616E75 73446562 75670D0A | 200 JanusDebug.. |
2 RK API 000000 : 30303031 38303032 44343336 3533320D | 00018002D436532. |
2 US Debug request from: 198.242.244.16 00018002D436532
2 RK API 000010 : 0A
2 RK APO 000000 : 32303020 362E312E 30472020 20362E39 | 200 6.1.0G 6.9 |
2 RK APO 000010 : 20363636 0D0A | 666.. |
18 RK API(2) 000000 : 48545450 2F312E30 20323030 20303030 | HTTP/1.0 200 000 |
18 RK API(2) 000010 : 31383030 32443433 36353332 0D0A | 18002D436532.. |
2 US *** Connection to worker: External user number: 21 Internal user number: 18

```

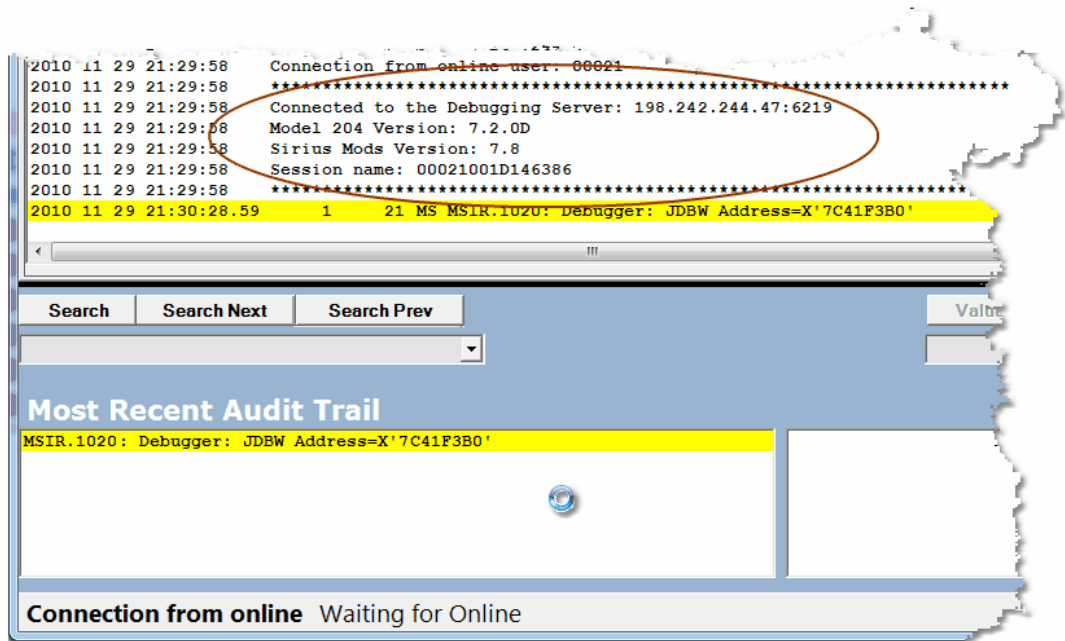
The worker thread reports its actions in lines that begin with three asterisks (***). The worker refers to the Online thread to be debugged as the "debuggee." A successful connection between the Debugger Server and Debugger Client results in a "Debugging is on" message, and the ID referred to in the connection handshake becomes the "session ID" for this debugging session:

```

2 US ***Worker got command from client: R
2 US ***Worker posts debuggee, and then waits...
18 MS MSIR.0915: Debugging is on: client is 198.242.244.16 port 8081, sessionID: 00018002D436532

```

- The worker thread is directed by the Debugger Client (based on user interface actions), and the worker thread maintains an internal master/slave, post/wait relationship with the Online thread that is being debugged. The initial action of the worker thread is to announce itself to the Debugger Client in the **Audit Trail** page:



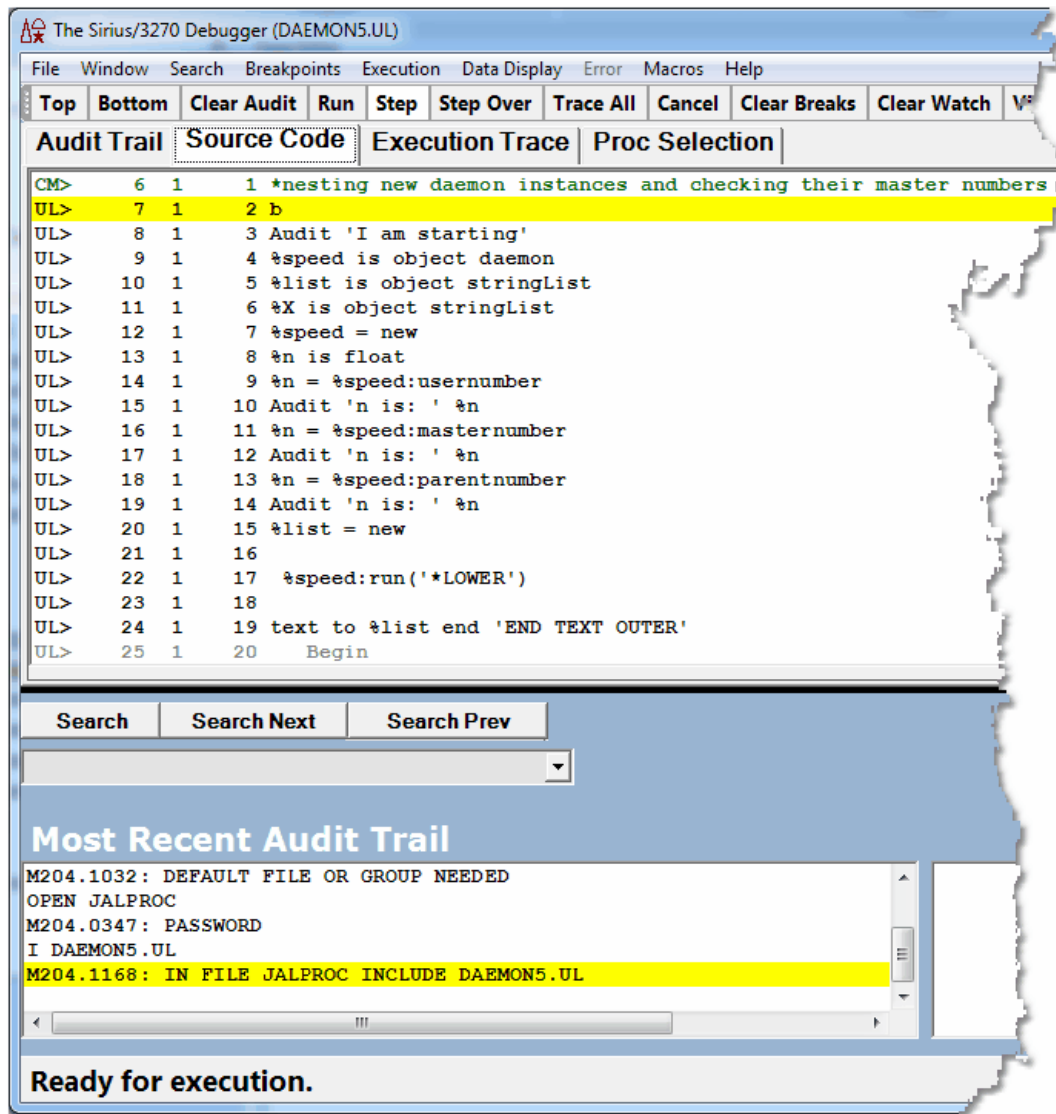
- The worker then waits for the Online thread to initiate debuggable activity. If you include a *Model 204* procedure from the Online thread, for example (DAEMON5.UL in the *SirScan* excerpt below), the worker detects this and pauses procedure processing while it sends the code lines to the Client (in an XML document) using the DEBCLI1 CLSOCK port, then waits for a response from the Client.

Note: The format of the XML messages in the following and subsequent code examples in this section is unpublished and subject to change.

```
18 LI I DAEMON5.UL
18 MS M204.1168: IN FILE JALPROC INCLUDE DAEMON5.UL
2 RK SCAN: US=SOCKUSER TM=C6F2F410 IP=198.242.244.16 JA=DEBUGSERVER3270
2 US ***Debugger Worker wakes up: S
2 US ***TraceCount: 0
2 RK API 000050 : 0A
2 RK APO 000000 : 3C446562 75674175 6469743E 3C737461 <DebugAudit><sta
2 RK APO 000010 : 74653E53 3C2F7374 6174653E 3C73746F te>S</state><sto
2 RK APO 000020 : 7054696D 653E3036 31393131 32333935 pTime>0619112395
2 RK APO 000030 : 3337303C 2F73746F 7054696D 653E3C6E 370</stopTime><n
2 RK APO 000040 : 6578743E 2D313C2F 6E657874 3E3C6C61 ext>-1</next><la
2 RK APO 000050 : 73743E2D 313C2F6C 6173743E 3C726574 st>-1</last><ret
2 RK APO 000060 : 75726E43 6F64653E 303C2F72 65747572 urnCode>0</retur
><<
```

```
2 RK APO 0001D0 : 20006173 74657220 6E756062 00 0000 master numbers
2 RK APO 0001E0 : 203C2F6C 696E653E 3C6C696E 653E4320 </line><line>C
2 RK APO 0001F0 : 20202020 20202020 20302020 20202020 0
2 RK APO 000200 : 32303320 20202020 20312062 20203C2F 203 1 b </
2 RK APO 000210 : 6C696E65 3E3C6C69 6E653E4D 533C2F6C line><line>MS</l
2 RK APO 000220 : 696E653E 3C6C696E 653E5353 20202020 ine><line>SS
2 RK APO 000230 : 20202020 20302020 20202020 32303420 0 204
2 RK APO 000240 : 20202020 20312041 75646974 20274920 1 Audit 'I
2 RK APO 000250 : 616D2073 74617274 696E6727 20203C2F am starting' </
2 RK APO 000260 : 6C696E65 3E3C6C69 6E653E53 20202020 line><line>S
2 RK APO 000270 : 20202020 20323820 20202020 20323035 28 205
2 RK APO 000280 : 20202020 20203120 25737065 65642069 1 %speed i
2 RK APO 000290 : 73206F62 6A656374 20646165 6D6F6E20 s object daemon
2 RK APO 0002A0 : 203C2F6C 696E653E 3C6C696E 653E5320 </line><line>S
2 RK APO 0002B0 : 20202020 20202020 32382020 20202020 28
2 RK APO 0002C0 : 32303620 20202020 20312025 6C697374 206 1 %list
2 RK APO 0002D0 : 20697320 6F626A65 63742073 7472696E is object strin
2 RK APO 0002E0 : 674C6973 7420203C 2F6C696E 653E3C6C gList </line><l
2 RK APO 0002F0 : 696E653E 53202020 20202020 20203238 ine>S 28
```

- The Client displays the procedure code in the Debugger **Source Code** page:



6. The Client user initiates the next round of activity by invoking an operation on the source code, say, stepping to the next statement. The Client sends this command in an XML message to the worker (upper circle, below), and the worker wakes up and instructs the Online thread. The worker reports a sequence of "****" audit lines (the ending **N**'s below mean "run to **N**ext statement" and "**N**ext statement executed"), then sends an XML response to the Client (lower circle, below) about the result of executing the next statement in the program:

```

2 RK API 000000 : 3C446562 75675265 71756573 743E3C6F | <DebugRequest><o
2 RK API 000010 : 70657261 74696F6E 3E4E3C2F 6F706572 | peration>N</oper
2 RK API 000020 : 6174696F 6E3E3C61 7267313E 3C2F6172 | ation><arg1></ar
2 RK API 000030 : 67313E3C 61726732 3E3C2F61 7267323E | g1><arg2></arg2>
2 RK API 000040 : 3C2F4465 62756752 65717565 73743E0D | </DebugRequest>
2 RK SCAN: US=SOCKUSER TM=C6F2F410 IP=198.242.244.16 JA=DEBUGSERVER3270
2 US ***Worker got command from client: N
2 US ***Worker posts debuggee, and then waits...
18 RK SCAN: US=JAL TM=JAL2
18 ST USERID='JAL' ACCOUNT='JAL' LAST='CMPL' SUBSYSTEM='' PROC-FILE='JALPROC' PRO
STBL=1553 UTBL=59 PDL=892 CNCT=13371 CPU=1 RQTM=13370903 SURD=1 SUWR=1 DKPR
2 US ***Debugger Worker wakes up: N
2 US ***TraceCount: 0
2 RK API 000050 : 0A
2 RK APO 000000 : 3C446562 75674175 6469743E 3C737461 | <DebugAudit><sta
2 RK APO 000010 : 74653E4E 3C2F7374 6174653E 3C73746F | te>N</state><sto
2 RK APO 000020 : 7054696D 653E3036 31393131 36323234 | pTime>0619116224
2 RK APO 000030 : 3436303C 2F73746F 7054696D 653E3C6E | 460</stopTime><n
2 RK APO 000040 : 6578743E 303C2F6E 6578743E 3C6C6173 | ext>0</next><las
2 RK APO 000050 : 743E2D31 3C2F6C61 73743E3C 72657475 | t>-1</last><retu
2 RK APO 000060 : 726E436F 64653E30 3C2F7265 7475726E | rnCode>0</return
2 RK APO 000070 : 436F6465 3E3C7661 6C756573 2F3E3C74 | Code><values><<t
2 RK APO 000080 : 72616365 20636F75 6E743D22 30222F3E | race count="0"/>
2 RK APO 000090 : 3C2F4465 62756741 75646974 3E0D0A | </DebugAudit>..

```

7. The Client-worker-Online communication continues in this fashion according to the commands invoked by the Client GUI user. The worker continues in a loop/dialogue with the Client, reporting its state and latest activity to the Client, and receiving XML requests from the Client (commands that are based on what the Debugger GUI user is invoking). The worker also maintains a master-slave relationship with the Online thread, guiding the execution of the program and reporting execution results.
8. In case you need to debug the Debugger, you can access the XML traffic exchanged between the worker and Debugger Client. To do so, use the JANUS TRACE command (described in the *Janus TCP/IP Base Reference Manual*) to increase the tracing on the Debugger Server port (in this example, DEBUGSERVER3270) and on the Online's client socket port (in this example, DEBCL11), and use *SirScan* to view the traffic. The JANUS DISPLAYTRACE command reveals the current trace values.

Since a high tracing value, say **15**, can capture huge amounts of data, remember to return the tracing settings to their former values when you no longer need so much detail.

Also of possible use in a debugging situation, the Debugger Client installation folder is the [default location](#)^[296] for a text log (**log.txt**) of Client activity and reference file information.

8.3 How the Janus Debugger handles communication breaks

If the *Janus Debugger* is not properly configured or if communication between any of its components is lost, the Debugger Client or the browser or both will display error messages and will attempt to restore normal operation, typically without requiring a recycling of any of the components.

In most cases, as long as the Debugger Client itself is not the problem, these communication breaks are signaled by the Client with a Communication Error message box and the display of **Communication Error** in the [Status bar](#)^[50].

This section describes the most likely types of communication breaks (not related to product configuration) and how the Debugger responds. The subsections below (except for the last) are organized by the Client's Communication Error message:

[Invalid response from debugger: info](#)^[354]

[An existing connection was forcibly closed by the remote host](#)^[356]

[The Debugger Client is not available](#)^[357]

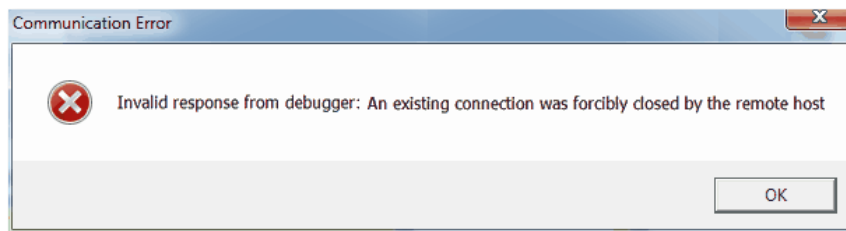
Invalid response from debugger: *info*

info is one of the following:

- **An existing connection was forcibly closed by the remote host**

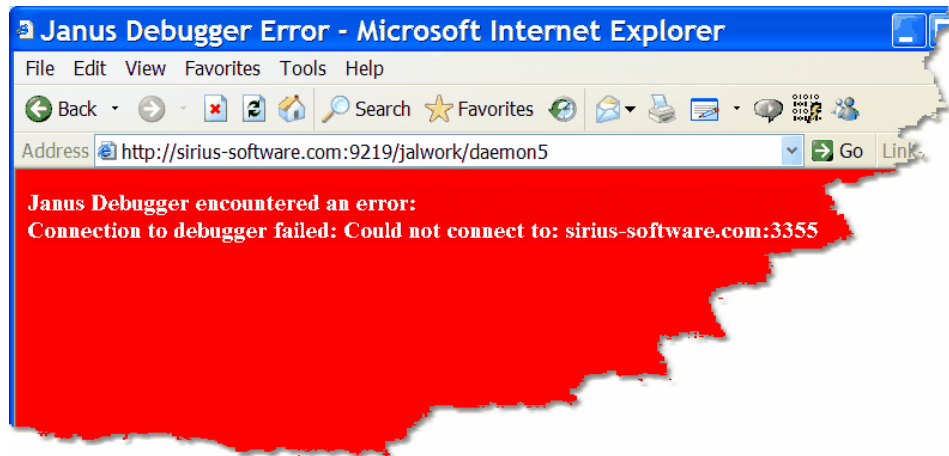
In this case, the network connection between the PC and the Online is lost, or the Debugger Server port is not started.

If the Debugger Client was in the process of debugging code, the next Client operation you attempt produces a Client message like the following, and **Communication Error** is displayed in the [Status bar](#)^[50]:



Clicking **OK** here removes the message, but processing cannot continue until the connection to the Online is reestablished or the Debugger port on the Online is restarted. There is no need to recycle the Debugger Client, although any program that was being processed is discarded.

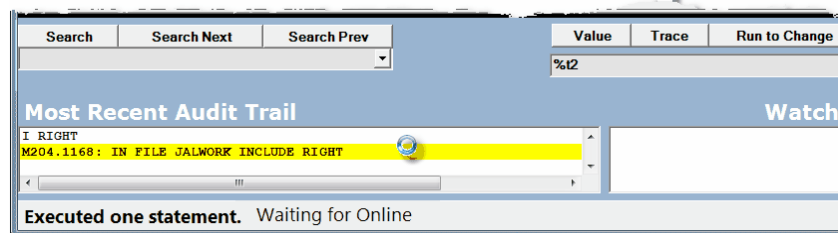
If the browser was waiting while the Debugger Client was processing, it continues to wait. If no debugging was in process, and the browser sends a new request to the Web Server, the browser is sent a reply from the Client like the following:



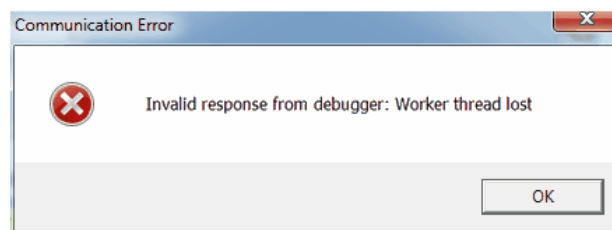
Once the broken connection is reestablished, debugging continues when the browser sends or resends a request. There is no need to recycle the browser.

- **Worker thread lost**

The Web Server or Debugger Server encounters a *Model 204* or *Janus* error while doing work for the Client, leaving the Client in an extended waiting state (**Waiting for Online** is displayed in the Status bar).



In such a case, the Server eventually times out (two minutes), and the Client displays **Communication Error** in the Status bar as well as an error message box like the following:



Shortly thereafter, the Client resumes processing, and the browser waits and follows its own timeout default.

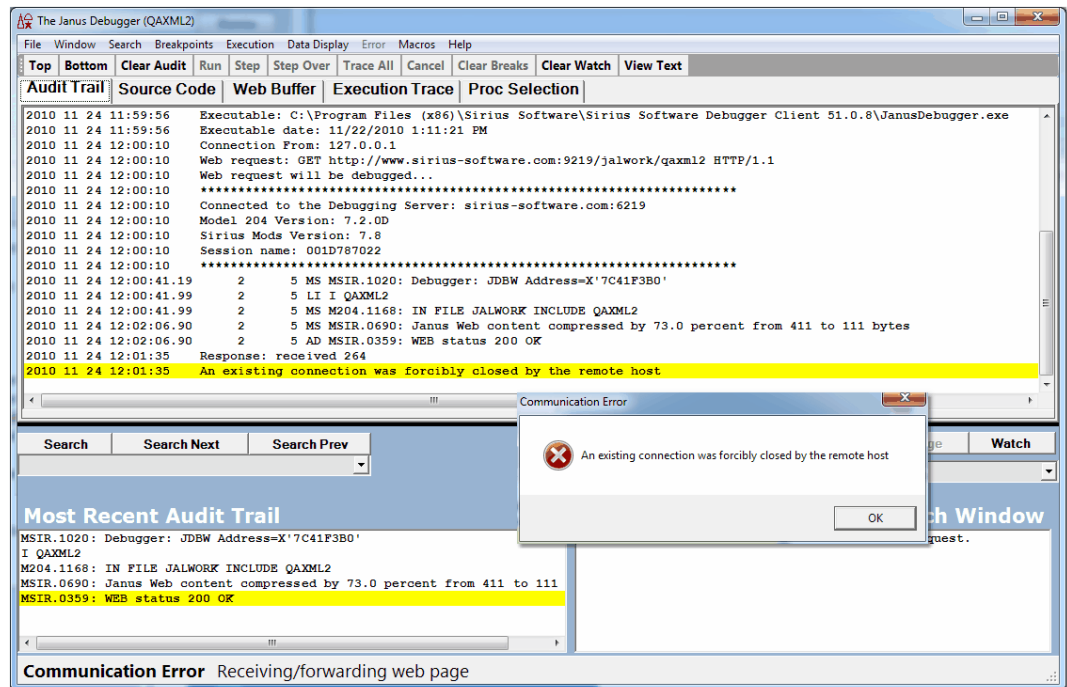
You can review the **Audit Trail** page for information to help locate the error in the Online processing.

An existing connection was forcibly closed by the remote host

Two cases of these browser problems follow:

- An external connection to the browser is broken.

As a new browser session begins, the Debugger processing is halted and a Communication Error message like the following is displayed:



In its role as the browser's proxy, the Debugger Client has received an error notification from a site that the browser has contacted, say to load an image, as part of its normal setup and home page presentation.

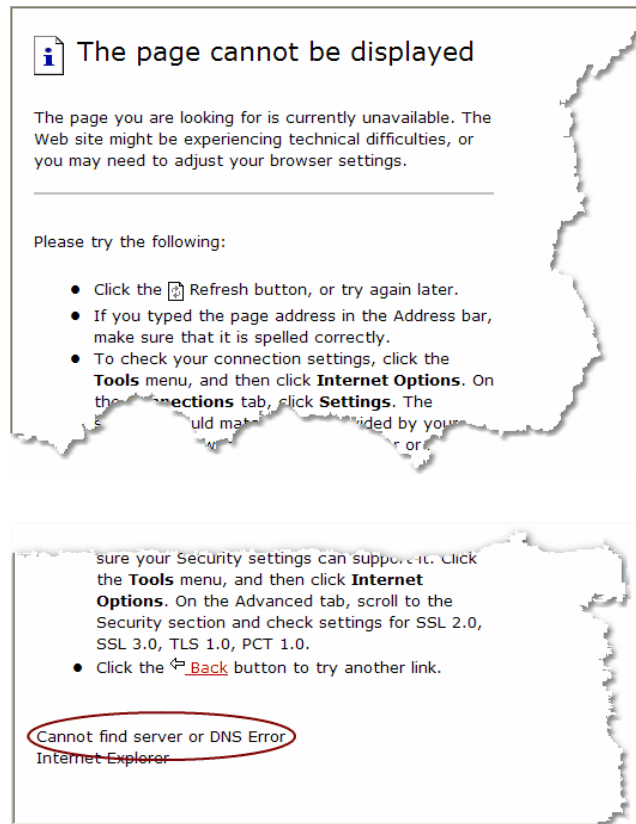
Such errors are infrequent and usually harmless, and clicking **OK** releases the Debugger to continue processing.

- The browser breaks the connection to the Debugger Client.

As the Debugger Client is processing code that a Janus Web Server sends in response to a browser request, a user sends another browser request (or the browser sends a scheduled polling request). The Client-browser socket gets broken, and the Client displays a message like the one shown above. When you click the message **OK** button, the Client-browser socket gets reestablished.

The Debugger Client is not available

Browser requests return a "cannot find server" or "proxy server is unavailable" message like the following when the Debugger Client goes down or is not started:



Once the Debugger Client becomes available, debugging continues when the browser sends or resends a request. There is no need to recycle the browser.

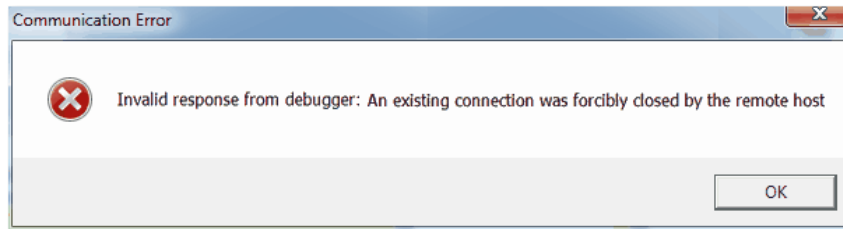
8.4 How the Sirius Debugger handles communication breaks

If the *Sirius Debugger* is not properly configured or if communication between the Client workstation and the host Online is lost, the Debugger Client will display error messages and will attempt to restore normal operation, typically without requiring a recycle of any of the components.

The most likely types of communication breaks (not related to product configuration) and the Debugger responses are described below.

The network connection between the PC and the host Online is lost, or the Debugger Server port is not started

If the Debugger Client was in the process of debugging code, the next Client operation you attempt produces a Client message like the following, and **Communication Error** is displayed in the [Status bar](#)⁵⁰:



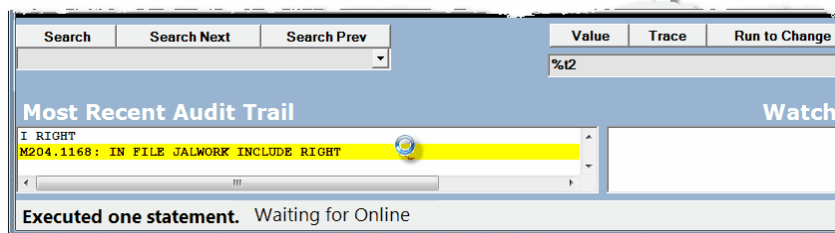
Clicking **OK** here removes the message, but processing cannot continue until the connection to the Online is reestablished or the Debugger port on the Online is restarted. There is no need to recycle the Debugger Client, although any program that was being processed is discarded.

If the Online was waiting while the Debugger Client was processing, loss of connection to the PC returns control to the Online user and causes the program to run to completion.

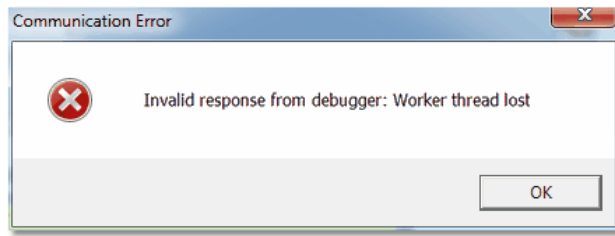
Once the broken connection is reestablished, debugging continues when the Online sends or resends a request. There is no need to recycle the Client.

Worker thread lost

The Web Server or Debugger Server encounters a *Model 204* or *Janus* error while doing work for the Client, leaving the Client in an extended waiting state (**Waiting for Online** is displayed in the Status bar):



The Server eventually times out (two minutes), and the Client displays **Communication Error** in the Status bar as well as the following error message box:



Shortly thereafter, the Client resumes processing.

You can review the **Audit Trail** page for information to help locate the error in the Online processing.

The Debugger Client goes down or is not started

Online requests run as normal when the Debugger Client is not available: that is, the program executes without first being intercepted by the Debugger.

Once the Debugger Client becomes available, debugging continues when the Online sends or resends a request.

If the Online is waiting while the Client was in the process of debugging code, a shutdown of the Client causes control to return to the Online user, and the code that was being debugged runs to completion in the Online.

8.5 Tracking Client performance

Programs with thousands of lines of code can introduce a slight delay while the Debugger Client reads, parses, and displays the code in the **Source Code** tab. As an aid to program tuning, you can have the Client log the time it spends processing program code. For example, you can isolate the elapsed time between the arrival of program code on the workstation and its display in the **Source Code** tab, ready for debugging.

To send code handling data to the Client log file (**log.txt**), you add an entry to the Client configuration file (**debuggerConfig.xml**). This file is installed in the same directory as the Debugger Client executable file, and it is [initially configured](#)^[374] as part of the Debugger Client installation.

To update the file:

1. Open the **debuggerConfig.xml** file in a text editor.

2. Add a `<collectTuningData/>` element at the same level (as a sibling of) the existing `<serverList>` element.

When complete, your configuration file should have a structure like the following:

```
<debuggerConfig version="1.0">
  <serverList>
    .
    .
    .
  </serverList>
  <proxy>
    .
    .
    .
  </proxy>
  .
  .
  .
  <collectTuningData/>
</debuggerConfig>
```

3. Save and close the file; then restart the Debugger Client.

After each subsequent Client interaction with the Online, the `log.txt` file will contain blocks of time-stamped lines like the following (the data is sent in XML documents, as described in [Debugging the Janus Debugger](#)^[345] and in [Debugging the Sirius Debugger](#)^[348]):

```
2008 10 03 10:07:20    Receive: read XML begin
2008 10 03 10:07:20    Receive: read XML end, bytes read=1819
2008 10 03 10:07:20    Receive: parse XML begin
2008 10 03 10:07:20    Receive: parse XML end
2008 10 03 10:07:20    Receive: process XML begin
2008 10 03 10:07:20    Incoming program state: S Prior state: I
2008 10 03 10:07:21    Receive: process XML end
2008 10 03 10:07:21    UI: load source begin
2008 10 03 10:07:21    UI: load source end
```

The last two sample lines above are recorded only when a new program is sent to the Client. Otherwise, all Client GUI commands provoke an XML message exchange between the Client and the Debugger Server that is reflected in a block of single pairs each of read, parse, and process lines.

8.6 Resolving issues when automatically maintaining IE proxy settings

The installation of the *Janus Debugger* calls for setting up the Debugger Client as a proxy server to intercept HTTP calls made from a browser. For sites using the Internet Explorer browser, an option in IE lets them designate certain exceptions, URL addresses that will *not* be routed to the proxy server. This option is the "Do not use proxy server for addresses beginning with:" box (accessed by **Tools > Internet Options > Connections > LAN Settings > Advanced**).

Although the Client has Preferences (**File menu > Preferences**) options to [automatically deactivate the proxy bypassing](#)^[385], the Debugger can be subject to issues such as hangs or timeouts when these exception URLs are invoked. The set of preconditions is:

- An IE proxy server is set up, and at least one exception URL is specified in IE
- Client **Preferences** settings include both of these:
 - The **proxy** option of **IE Mode**
 - **Clear IE proxy override** in the **IE Options** area

The recommended initial remedy for these kinds of problems is a Client **Preferences** option that routes these exception URLs through the Debugger Client to the proxy server IE was using before the Debugger installation. The **Preferences** option is **Use existing IE proxy for URLs not to be debugged** (in the **IE Options** area). As a result, the Client will collect troubleshooting information concerning its handling of these URLs, and it will display messages in its **Audit Trail** page:

```
2010 06 08 08:01:46    Web request: GET http://www.google.com/ HTTP/1.1
2010 06 08 08:01:46    Web request will NOT be debugged:
www.google.com:80 not listed in the configuration ...
2010 06 08 08:01:46    Passing thru not debugged request:
http://www.google.com/ to proxy: 127.0.0.1:808
```

CHAPTER 9 *Installation and Configuration*

This section describes how to install the *Janus Debugger*, the *Sirius Debugger*, or both. Once you complete this installation, you can begin to use either or both products.

The working part of this section consists of two subsections of mostly cookbook-style directions. It is recommended that you first complete the steps in the "Online Configuration" section, then complete the "Workstation Configuration" section.

Most of the steps are to be performed whether you are installing one or both Debugger products, and exceptions are noted.

If you will be using a local editor (*Xtend* or *UltraEdit*) with the Debugger, finish the Debugger installation first, then see [Using a local editor](#)^[166].

For information about setting up at your site a centralized distribution of updated Debugger Client replacement files, see [Providing updated versions of the Debugger Client](#)^[390].

9.1 Overview

To use the *Janus Debugger* or the *Sirius Debugger*, you must first install and configure two components:

Debugger Server	Located in the Online for which Janus Web or 3270/Batch2 threads are to be debugged, this is a <i>Janus Sockets</i> server socket application. The Server manages the “worker” threads used by the Debugger to control the threads that are being debugged.
-----------------	---

Installing and configuring this Server is done once for each *Model 204* Online whose web or 3270/Batch2 threads are to be debugged.

Client GUI	Located on each workstation where the Debugger is to be used. The Client acts as an intermediary between the developer’s Web browser or output terminal and the online, and it provides the user interface for controlling the Debugger.
------------	--

Installing and configuring this Client is done once for each workstation.

The *Model 204* Online portion of the installation should be done first, since decisions made doing the Online setup are input to the Client setup (for example, the port number of the debugging server). Because there are two installation locales, each of which will often be done by different people in an organization, the installation instructions are divided into two “tracks”:

Online Configuration	Performed by <i>Model 204</i> system manager or equivalent.
-----------------------------	---

Workstation Configuration	Most likely performed by individual developers on their own machines, after the Online configuration is complete.
----------------------------------	---

Sirius recommends that the system manager do one workstation configuration after completing the online configuration, verifying that the installation and configuration were successful.

9.2 Online Configuration

This section shows how to prepare a *Model 204* Online for use of the *Janus Debugger*, the *Sirius Debugger*, or both.

The installation tasks require *Model 204* system manager privileges. Also, since you must set at least one User 0 parameter, a cycle of the *Model 204* Online will be required.

The Online configuration tasks are listed below.

[Check prerequisites](#)^[365]

[Authorize the Debugger](#)^[365]

[Set Model 204 system parameters](#)^[366]

[Define and start the Debugger Server port](#)^[367]

[Define and start a client socket port \(Sirius Debugger only\)](#)^[368]

9.2.1 Check prerequisites

Do not continue until the following product version requirements have been met:

1. For the *Janus Debugger* or *Sirius Debugger*, make sure that you are running Version 7.0 or higher of the *Sirius Mods*, and at least *Model 204* Version 6.1.

You can verify this by issuing the **SIRIUS** command at the *Model 204* command prompt, for example:

```
Site - SIRIUS                      Job name ULSPFPRO
Sirius Mods version - 7.8         Model 204 version 7.2.0D
```

2. A *Janus SOAP* license is required to use the [DebuggerTools class methods](#)^[161].

9.2.2 Authorize the Debugger

1. Through your Sirius Sales representative:
 - a. Arrange a Janus Debugger Trial or Sirius Debugger Trial agreement for one or more seats of the *Janus Debugger* or *Sirius Debugger*.

The number of “seats” or connections indicates the number of programmers who will be able to use the Debugger concurrently.

- b. Obtain information about applying the Authorization Key that enables the appropriate software.

You may need to download a key from the Sirius Software web site. The “Customer Service” page (<http://sirius-software.com/support.html>) contains a link to download an authorization zap and also contains information about the required site user ID and password.

2. Once authorization is complete, view the output of the **SIRIUS** command in your Online to verify that you have some authorized Debugger seats. The output should contain one or both of lines like this:

Janus Debugger Expires 02/24/2009 Max connections 10

Sirius Debugger Expires 02/24/2009 Max connections 3

9.2.3 Set Model 204 system parameters

User 0 parameters

1. Set the *Model 204* User 0 parameter **DEBUGMAX**. This parameter, which defaults to 0, is the number of internal debugging control blocks that will be allocated.

Specify a **DEBUGMAX** value of at least the number of seats authorized by the Authorization Key (or the sum of the numbers of seats, if keys for both Debuggers) you obtained in the previous step. You can set it higher, however, if you anticipate getting a key in the future that adds seats.

Note: The maximum number of concurrent debugging sessions may never exceed the seat count of the key(s), no matter what value you specified for **DEBUGMAX**.

2. Set the User 0 parameter **DEBUGPAG**, if necessary. This parameter specifies the upper limit of the **CCATEMP** page allowance for a debugging session (for the combination at any one time of Audit Trail and Source Code data). The Debugger uses **CCATEMP** pages for its audit trail and source code lines.

DEBUGPAG defaults to 100.

If you are going to debug large programs (more than 1000 lines in a single request), increase the **DEBUGPAG** setting to, say, 250. Its maximum is 25000.

Note: Although **DEBUGPAG** is resettable by the system manager, **DEBUGMAX** is **not** resettable.

3. The **SDAEMDEV** User 0 parameter sets the **IODEV** number for the Sirius sdaemon threads that are the Debugger Server worker threads. In any Online where either the *Janus* or *Sirius Debugger* will be used, these threads require a pushdown-list size (*Model 204* **LPDLST** parameter setting) of at least 9000.

Note: As described below, the PDL size is user-resettable, and the *Model 204* thread that is running the program that is being debugged also uses additional PDL space.

4. The User 0 parameters SDEBGUIP and SDEBWRKP set the default values respectively of the [workstation port number](#)^[370] on which the Debugger Client is listening, and the port number in your Online that [is defined](#)^[367] for worker threads. Explicitly setting these parameters is optional.
5. Cycle the Online, so the parameter settings take effect.

User parameters

1. As stated above, the Debugger Server worker sdaemon threads require a pushdown-list size of at least 9000.
2. A thread that runs a request to be debugged under the Debugger will use more PDL, VTBL, and STBL space than normal. If you suspect the existing values at your site for the size of these areas may not be adequate, add 2000 bytes to the the PDL size, about 100 bytes (3 units) to VTBL, and about 300 bytes to STBL. You can use `UTABLE LVTBL`, `UTABLE LSTBL`, and `UTABLE LPDLST` commands.

9.2.4 Define and start the Debugger Server port

1. Select an unused TCP port number on which your Online will run the Debugger Server.

A Janus server socket port will be opened on that port number to service requests for Debugger Server worker threads. Make sure that port number is free of any security restraints at your site that might prevent access by any Debugger Client workstation seeking a connection.
2. Confirm that, as described in [Set Model 204 system parameters](#)^[366], the Debugger Server *worker threads* have a *Model 204* LPDLST parameter setting of at least 9000 in any Online where either the *Janus* or *Sirius Debugger* will be used.
3. Define and start a Debugger Server port.

You may want to consult with your system manager or whomever sets up Janus port definitions for your site. The ***Janus Sockets Reference Manual*** documents port definition particulars.

For Debugger purposes, the JANUS DEFINE command format is:

```
JANUS DEFINE portName portNumber DEBUGGERSERVER maxConnections
```

Note the DEBUGGERSERVER port type, which is a Debugger-only, license-free, Janus server socket.

An example command sequence follows, which creates/recreates and starts a server port on port 3226, then displays the full port definition:

```
JANUS DRAIN DEBUGSERVER3226
```

```
JANUS DELETE DEBUGSERVER3226
```

```
JANUS DEFINE DEBUGSERVER3226 3226 DEBUGGERSERVER 15 TRACE 0
```

```
JANUS START DEBUGSERVER3226
```

```
JANUS DISPLAY DEBUGSERVER3226
```

For your convenience, you may want to store this command sequence for reuse in a *Model 204* procedure.

4. Issue the following command to check that the Server port started:

```
JANUS STATUS
```

You should see a line like the following (using the naming convention in these examples and procedures):

```
DEBUGSERVERportNumber portNumber sockType Start statistic-values
```

Where *portNumber* is the port number from above, and *sockType* is DEBUGSRV.

5. Once you know the Debugger Server has started, the best way to verify its functioning is to install the Debugger Client on a workstation and make sure the workstation can connect, as described in [Workstation Configuration](#)^[369].

9.2.5 Define and start a client socket port (Sirius Debugger only)

To initiate *Sirius Debugger* sessions from this Online, you must have a Janus client socket port to connect to the Debugger Client that is running on the workstation. You may want to consult with your system manager or whomever sets up Janus port definitions for your site, and the ***Janus Sockets Reference Manual*** documents the port definition particulars.

A sample Janus port definition (for the port DEBCLIENT7) follows. The DEBUGGERCLIENT port type is a Debugger-only, Janus client socket that requires a Debugger license.

```
JANUS DEFINE DEBCLIENT7 * -
  DEBUGGERCLIENT 5 REMOTE * * -
  LINEND 0D0A -
  TRACE 0 -
  TIMEOUT 40 -
  MASTER -
  SOCKPMAX 1

JANUS START DEBCLIENT7

JANUS CLSOCK DEBCLIENT7 ALLOW
```

Notes:

- If you have a Debugger license and a *Janus Sockets* license, you can use a CLSOCK port type instead of DEBUGGERCLIENT.
- The JANUS DEFINE parameters KEEPALIVE and SSL are valid for CLSOCK ports but not for DEBUGGERCLIENT ports or for a CLSOCK port used for the Debugger.
- The JANUS CLSOCK ALLOW statement applies to both CLSOCK and DEBUGGERCLIENT ports. As specified above (with no additional parameters), unrestricted access is allowed to the port.

9.3 Workstation Configuration

This section specifies how to prepare a workstation for use by the *Janus Debugger*, *Sirius Debugger*, or both.

These are the workstation configuration tasks:

Perform preliminary tasks^[369]

Run, check, and verify the Client installation^[371]

Customize the debuggerConfig.xml file^[374]

Configure the web browser (Janus Debugger only)^[380]

Test the end-to-end configuration^[387]

9.3.1 Perform preliminary tasks

1. Make sure your workstation environment is adequate for the Debugger Client:

- The operating system must be Windows XP or Windows 7. Windows 2000 is also acceptable for builds of the Client prior to 51.
- The target drive of the workstation must have at least fifteen megabytes of free disk space.
- The Microsoft .NET Framework (3.5 SP1) must be installed on your machine. Version 3.5 is included with Windows 7, and it can also be installed on Windows XP.

If .NET 3.5 is not on your machine at the time of Client installation, the installer program will **not** complete, and it will inform you to download and install the Framework. You can get the Framework from:

<http://msdn.microsoft.com/en-us/netframework/cc378097.aspx>

Or locate the download from:

<http://msdn.microsoft.com/en-us/netframework/default.aspx>

- For *Janus Debugger* sessions, your browser must support proxy servers.
Most, if not all, current popular browsers have this support. Consult your browser's documentation or "properties" information. The "[Configure the web browser](#)"^[380] section describes how to set up a proxy for several popular browsers.
 - It is assumed you are accessing the Debugger Client locally, that is, from the machine on which the software is installed. Accessing the Debugger Client and its online Help file from another workstation may be problematic. Microsoft security may prevent you from accessing the Debugger compiled Help file from a networked location. For more information, contact Sirius Support.
2. Obtain the following information from the system manager of the Online running the web or 3270/Batch2 threads that are to be debugged:
- Confirmation that the Debugger Server is running
 - The host name (or IP number) of the Online
 - The port number of the Debugger Server
 - For *Janus Debugger* sessions, the port number on that Online of the web server whose threads are to be debugged
 - For *Sirius Debugger* sessions, the name of the client socket port [set up](#)^[368] in that Online for the programs that are to be debugged
3. On this workstation, select a local TCP port number for the Debugger Client to listen on, if the default (8081) is not available.
- For *Janus Debugger* sessions, the Client acts as a proxy server between the local web browser and the Janus Web Server; its "address" is *localhost:port*, where *localhost* is the workstation machine's host name or IP address, and *port* is the listening port number. For *Sirius Debugger* sessions, the workstation port is for communication between the Debugger Client and *Model 204* Online.
- If you are already using port 8081 for something else, or if you cannot use 8081 for some other reason, you must select a new port number. Port numbers in the 8000 range are often used for proxies, but any unused port number is acceptable.
- For information about the TCP ports that are currently active on your workstation, you can issue the NETSTAT command from an MS DOS command box on your machine.
- If you are going to change this port number, have the new number ready when you [customize the debuggerConfig.xml file](#)^[374], later.
4. For *Sirius Debugger* sessions, determine the host identification of the workstation where the Debugger Client will run.
- This may be either an IP number or DNS name. Two ways to obtain the IP number are:
- Select the **Audit Trail** tab in the Client main window, and click the **Top** button. Approximately the fourth line from the top displays the **Local IP address** of the Client workstation.

- Open an MS DOS box, and issue the `IPCONFIG` command. You will see output similar to this:

```

Ethernet adapter ....
    Connection-specific DNS Suffix  . :
    IP Address. . . . . : x.x.x.x   <== you want this
    Subnet Mask . . . . . : y.y.y.y
    Default Gateway . . . . . : z.z.z.z

```

9.3.2 Run, check, and verify the Client installation

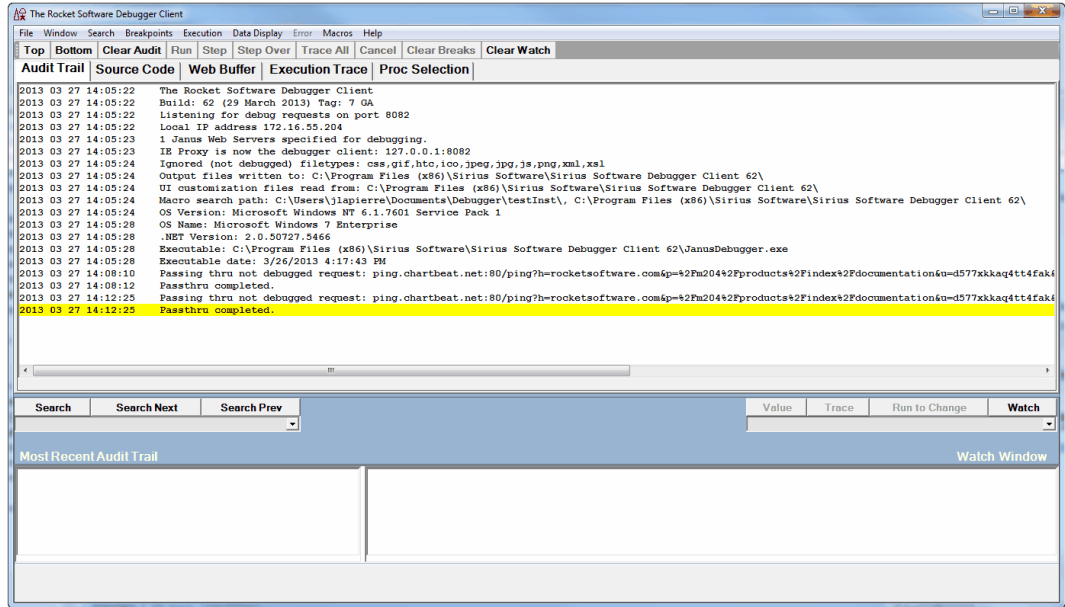
1. To install or upgrade the Debugger Client, download, extract, and run the SETUP.EXE file, which invokes a Windows installer, using either of the the following links:

- <http://www.sirius-software.com/debclient.zip>
- The "Download the ZIP file" link you reach via the "Download client (workstation) files" link in your individual Customer Maintenance Area — reached from <https://sirius-software.com/maint/clist> — on the Sirius Software web site.

If you are upgrading to a newer build of the Client, the installer program will **not** remove the previous Debugger Client configuration file (`debuggerConfig.xml`).

If you are re-installing the same build version of the Client, the installer insists that you first remove the existing Client files (using the Add/Remove program from the Windows Control Panel). The uninstaller program will **not** remove the Client configuration file (`debuggerConfig.xml`), and the installer will not overwrite it.

- After the installer finishes, double-click the **Debugger Client** icon (from the Windows system **Start** menu or from the desktop). The Client should open, displaying a list of messages that identify the Client and its host, as well as key file locations and software versions; the last message is yellow-highlighted:



- Make sure the installer did the following:



- Placed an icon on your desktop to run the Debugger Client:
- Created a Windows **Start Menu** item under:

Programs > Sirius Software > Sirius Software Debugger Client

For Client builds prior to 50, the target folder name is **Client for Janus and Sirius Debuggers**.

- Installed these files in the target installation folder:

JanusDebugger.exe	The actual Debugger Client program.
updateGet.exe	Program for downloading updated versions ^[390] of the Client.
unzip.exe	Freeware program for unpacking zipped files.
debuggerConfig.xml	XML-based configuration file that sets startup parameters for the Debugger Client. You must tailor this file, as directed in the next step.
Org.Mentalis.Security.dll	Support for SSL connections.
msvcr71.dll	Microsoft C library functions used by .NET programs.
gdiplus.dll	Microsoft graphics file.
whitelist.txt	White List ^[79] file template.
dir.txt	Empty file installed to test the writability of the folder(s) designated ^[378] to contain various Client work files.
JanusDebugger.chm	A Windows compiled HTML Help file. To view the Help, double-click the .chm file or access it from the Debugger Client GUI Help menu.
jdebugr.pdf	<p>The Janus/Sirius Debugger User's Guide, which includes installation instructions and a copy of the online Help contents, formatted for convenient printing.</p> <p>To display PDF files, you need the Adobe Acrobat Reader, which you can download from http://www.adobe.com/products/acrobat/.</p> <p>For Client builds prior to 50, the jdebugr.pdf file is installed in the Doc installation subfolder.</p>

4. [Set up](#)^[374] the Debugger Client configuration file.

9.3.3 Customize the Debugger configuration file

After you install the Debugger Client for the *Janus Debugger*, you must modify the Client configuration file (`debuggerConfig.xml`) as described in the first six steps below. This file is installed in the same directory as the Debugger Client executable file. It can be accessed for editing via the Client's **File** menu.

If you are using the Client only for the *Sirius Debugger*, begin with [step 5](#)^[376].

If this is an initial installation of the Debugger and you are primarily interested in getting the Client up and running, you may ignore for now and return later to view the [Additional configuration options](#)^[378], below.

The configuration file contains parameters that associate the Debugger with the *Model 204* online(s) whose Janus Web threads you want to debug. A template `debuggerConfig.xml` file is placed in your client installation target folder along with the executable for the Debugger Client (`JanusDebugger.exe`). The template file is amply annotated with XML-style comments which describe the contents.

1. As supplied, `debuggerConfig.xml` contains a single `serverList` element (bounded by the `<serverList>` start tag and the `<\serverList>` end tag) that has a single `server` sub-element. A `server` element defines a Janus Web Server whose User Language requests you can debug.

Provide within the `serverList` element a `server` element for each Web Server with which you want to use the *Janus Debugger*. Simply copy and paste the supplied `server` element.

For example, if you are debugging for two Web Servers, the structure of your `serverList` element should be like this:

```
<serverList>
  <server>
    <host></host>
    <webPort></webPort>
    <workerPort></workerPort>
  </server>

  <server>
    <host></host>
    <webPort></webPort>
    <workerPort></workerPort>
  </server>
<\serverList>
```

- For each **server** element within **serverList**, provide values for the three entries shown in the **Field** column below. You obtained the appropriate values [earlier](#)^[369]. Specify the values between the pairs of angle-bracketed tags, as shown in the **Comment** column below:

Field	Comment
<code><host></host></code>	<p>The identifier of the machine that hosts the <i>Janus Web Server</i> application for which requests will be debugged. Either a TCP/IP host name or an IP Number may be specified. For example: <code><host>sirius-software.com</host></code>.</p> <p>Note: If you use an IP Number in this configuration file to identify a host, you must use the IP number when you reference a URL in your web browser. If you use a DNS name in this configuration file, you must also use it from the browser.</p>
<code><webPort></webPort></code>	<p>The port on this host that runs a <i>Janus Web Server</i> whose threads you want to debug. For example: <code><webPort>3224</webPort></code>.</p>
<code><workerPort></workerPort></code>	<p>The port on which this host provides <i>Janus Debugger</i> worker threads. These are the server socket threads that control the thread being debugged. They are set up^[367] as part of your <i>Model 204</i> Online configuration for the Debugger. For example: <code><workerPort>3226</workerPort></code>.</p> <p>Note: The worker port must be defined in the same Online that runs the Web Server identified by the values for host and webPort. If you are debugging multiple Web Servers in the same Online, they may all use the same worker port.</p>

When an incoming HTTP request is received by the Debugger Client, its host and web port are extracted and matched against those of the servers in the **serverList** element. This matching is case insensitive, but otherwise it must be an exact match.

If you specify an invalid value (for example, a non-numeric port number), the Client will fail to start and you will receive an "Error reading the debuggerConfig.xml" message box.

- If a Web Server you are debugging is secured (Janus Network Security), you must direct the Debugger Client to connect to the Web Server using the Secure Sockets Layer (SSL) protocol. Add an empty **ssl** sub-element to the appropriate **server** element defined in the previous step.

For example:

```
<server>
  <host>sirius-software.com</host>
  <webPort>3224</webPort>
  <workerPort>3226</workerPort>
  <ssl/>
</server>
```

4. By default, when the *Janus Debugger* examines a web request a browser sends, it excludes URLs with the following file types from debugging:

css	jpeg	png
gif	jpg	xsl
htc	js	xml
ico		

Each such skipped URL produces a "Web request will NOT be debugged" message in the Client's audit trail.

You may replace (entirely) the default list of file types, above, with a list you specify by adding a **filter** element to the **debuggerConfig.xml** file.

In the **filetype** subelement of the **filter** element, you specify (without regard for case and without a leading period) only the file types that will *not* be debugged, and these types replace the default list.

The file type filter applies to all web servers you debug, so the **filter** tag is a child of the root in the **debuggerConfig.xml** file. For example:

```
<debuggerConfig version="1.0">
  <serverList>
    ...
  </serverList>
  ...
</editor>
  ...
  <filter>
    <filetype>jpg</filetype>
    <filetype>xml</filetype>
  </filter>
</debuggerConfig>
```

Note: If your browser is Internet Explorer, you can also [filter by server host](#)^[380] the web requests to be debugged.

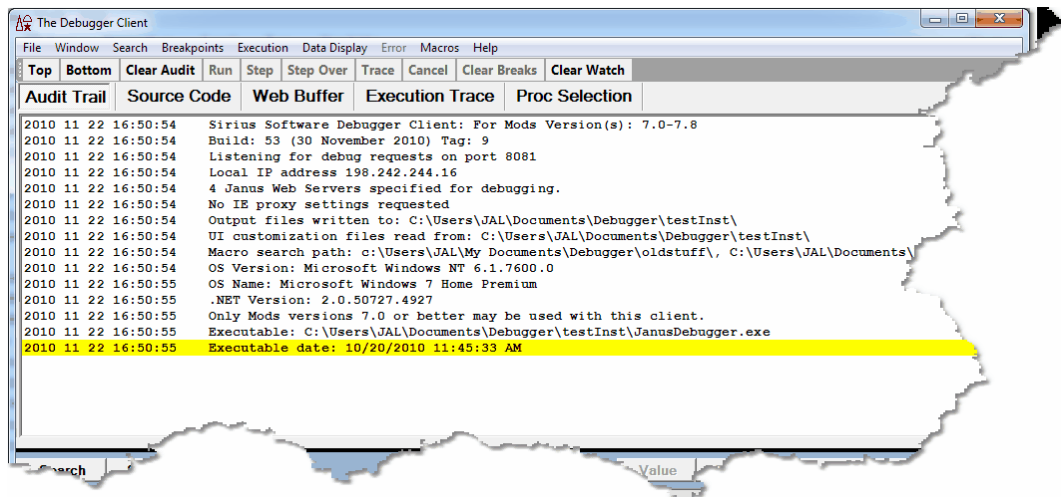
5. As discussed in the [preliminary tasks](#)^[369], the Debugger Client requires a host workstation TCP port on which to listen during *Janus Debugger* or *Sirius Debugger* sessions. If the default port number, 8081, is not available on your workstation, specify a suitable value for the **port** field in the **proxy** element of the **debuggerConfig.xml** file.

For example:

```
<debuggerConfig version="1.0">
  <serverList>
    ...
  </serverList>
  <proxy>
    <port>8888</port>
  </proxy>
  <editor>
    ...
  </editor>
  ...
</debuggerConfig>
```

- Save your changes to the `debuggerConfig.xml` file, then restart the Debugger Client (either select **Restart** from the Client's **File** menu, or exit and run the Client again). The changes to the `debuggerConfig.xml` file do **not** take effect until the Debugger Client is recycled.

The Client should open, ready for debugging, with information lines like the following:



For *Janus Debugger* sessions, the message report includes the [number of Web Servers](#)^[147] for which the Client is configured and whether the Internet Explorer browser settings, if any, will be [automatically toggled](#)^[382] by the Debugger. For *Sirius Debugger*-only sessions, the highlighted message will be the same as seen [earlier](#)^[371], reporting that the *Janus Debugger* is not active.

Note: Your modified `debuggerConfig.xml` file is **not** overwritten if you rerun the Debugger Client setup.exe file, for this or any subsequent versions of the Client.

Additional configuration options

The following options are separated from the preceding steps because they are not essential to an initial installation of the Debugger.

Alternative locations for the Client work files

During your debugging session, the Client stores information in various text files it creates or expects to find, by default, in the same folder as the Debugger Client executable file. If this folder location is inconvenient, three optional elements in the `debuggerConfig.xml` file let you specify alternative locations for the Client work files. You can add these root-child elements to `debuggerConfig.xml`:

- `<stateFileFolder>` specifies where most Client work files are written (log, preferences, searches, for example)
- `<macroLibraryFolder>` specifies where macro files are stored
- `<uiFolder>` specifies where the Client UI customization settings (`ui.xml`, `uimore.xml`) are stored

For example:

```
<debuggerConfig version="1.0">
  <serverList>
    ...
  </serverList>
  ...
  <stateFileFolder>c:\myData</stateFileFolder>
  <uiFolder>c:\myUI</uiFolder>
  <macroLibraryFolder>c:\work\macroLibrary</macroLibraryFolder>
  ...
</debuggerConfig>
```

For more information about the Client files that are stored in these locations, see [Changing the location of Client work files](#).^[296]

Font size

The font size in most Client windows is [scalable](#)^[299] by specifying a valid setting of the `fontScale` element. For example:

```
<debuggerConfig version="1.0">
  <fontScale>1.33</fontScale>
  <serverList>
    ...
  </serverList>
  ...
</debuggerConfig>
```

Window transparency

The degree of transparency of Preferences and [external-button](#)^[42] windows is adjustable by specifying a valid setting of the `opacity` element (max transparency is .01; max opacity is 1; default is .9). For example:

```
<debuggerConfig version="1.0">
  <opacity>.75</opacity>
  <serverList>
    ...
  </serverList>
  ...
</debuggerConfig>
```

Information URLs

The destination URLs of the **Sirius Documentation** and **Sirius Wiki** links in the Debugger Client's **Help** menu default respectively to <http://sirius-software.com/maint/manlist> and <http://m204wiki.rocketsoftware.com>. If your Internet access is limited, you can override these defaults by specifying the `documentationURL` and `wikiURL` elements. For example:

```
<debuggerConfig version="1.0">
  <documentationURL>http://myDomain.net/SiriusManuals</documentationURL>
  <wikiURL>http://123.4.5.666/mySiriusWiki</wikiURL>
  <serverList>
    ...
  </serverList>
  ...
</debuggerConfig>
```

These must be absolute URLs, and the host domain specified may be a DNS name or an IP number.

Text file editor

Many Debugger tasks entail the editing of small text files, and typically the Client opens by default the Microsoft *Notepad* editor (`notepad.exe`), which is guaranteed to be present on a Windows system. You can change default editors by specifying a suitable value in the `notepadReplacement` element.

For example, the following defines *Notepad++* (<http://notepad-plus-plus.org/>) as the default text editor:

```
<debuggerConfig version="1.0">
  <notepadReplacement>C:\Program Files\Notepad++\notepad++.exe</notepadReplacement>
  <serverList>
    ...
  </serverList>
  ...
</debuggerConfig>
```

The element value must be a full operating system (DOS style) path that identifies the executable file. An eligible editor must take the file to be edited as its first command line parameter. Given the definition above, for example, and selecting the **Edit debuggerConfig.xml** option from the Client's **File** menu, directs the Client to execute the following:

```
C:\Program Files\Notepad++\notepad++.exe debuggerConfig.xml
```

Multiple Client configurations

If, for testing purposes, you want to have multiple configurations, you can specify a configuration file other than `debuggerConfig.xml` on the command line:

1. Open an MS DOS command line window.
2. Navigate to the Debugger installation folder (it contains `JanusDebugger.exe`)
3. At the command line, issue `JanusDebugger myconfig`. This starts the Debugger using the file specified by `myconfig`, instead of `debuggerConfig.xml`. For example:

```
JanusDebugger test1.xml
```

If you are going to make multiple configuration files, start by cloning the `debuggerConfig.xml` file that is installed by default. This will save work and minimize the chance of errors.

Startup macro

A Debugger [macro](#)^[309] lets you automate a variety of Client tasks. You can invoke a macro in various ways, one of which is automatically, whenever the Client is started. To invoke such an automatic running of a macro at Client startup, you specify the macro name as the value of the `startup` attribute of the top level tag of the `debuggerConfig.xml` file. For example:

```
<debuggerConfig version="1.0" startup="startup.macro">
```

9.3.4 Configure the web browser (Janus Debugger only)

Set up your web browser to use the Debugger Client as a proxy server. Information follows for these individual browser types:

[Chrome](#)^[381]

[Firefox](#)^[382]

[Internet Explorer](#)^[382]

[Lynx](#)^[386]

[Opera](#)^[387]

You need the Janus Proxy port number you determined earlier in the [preliminary tasks](#)^[370] section. The default port is 8081, but you can change this by [editing](#)^[374] the `debuggerConfig.xml` file.

Once you define a proxy server for a browser:

- The Debugger Client must be running in order to operate that browser.
You will need to turn off the proxy in order to use the browser without the Debugger Client. Such toggling of the proxy definition is a Client option [available](#)^[383] for the Internet Explorer browser (only). Otherwise, it may be more convenient to point one browser at the Debugger Client proxy and use a different browser for your other Internet access.
- While you are debugging, you can use a second tab in your browser to access the Internet while another tab is occupied with User Language debugging. *Secured (HTTPS) connections are **not** allowed*. The occupied tab, the tab that invoked the Debugger, continues to remain unavailable for the duration of each debugging session.

Note: If it is not possible or not desirable to change the proxy settings on your web browser, an [alternative way to debug web threads](#)^[157] is to invoke the Debugger from a command you insert in procedures you run on Janus Web threads.

Chrome

To set up the proxy for Chrome browsers:

1. At the right end of the bar that contains the Omnibox navigation control, open the customization menu by clicking the three-stacked-lines icon:



2. In the menu, select **Settings**.
3. At the bottom of the the **Settings** tab, click the **Show advanced settings...** link.
4. Find the **Network** section, then click the **Change proxy settings...** link.

For Windows users, this opens the **Internet Properties** (or **Internet Options**) dialog box. The following steps are the same as those described below for the Internet Explorer browser:

- a. Select the **Connections** tab, and click the **LAN Settings** button.
- b. Locate the **Proxy Server** area, select the "Use a proxy server" checkbox, and provide the following:
 1. In the **Address** box, enter `localhost`.

2. In the **Port** box, enter the proxy listening port number discussed in the introduction above.
- c. Click **OK**.

[Click this link^{\[383\]}](#) to go to the Internet Explorer section and continue with further proxy set-up options.

Firefox

To set up the proxy for Firefox browsers:

1. From the **Tools** menu, select **Options > Advanced > Network**.
2. In the **Connection** box, click **Settings**.
3. In the **Connection Settings** dialog box, select **Manual proxy configuration**.
 - a. In the **HTTP Proxy** box, specify **localhost**.
 - b. In the adjacent **Port** box, specify the port number discussed in the introduction above.
 - c. Use the **No Proxy for** box to specify domain names that the browser will access directly, that is, for which the Debugger proxy will be bypassed.
4. Click **OK**.

Internet Explorer

To set up the proxy manually:

1. From the **Tools** menu, select **Internet Options**; then select the **Connections** tab, and click the **LAN Settings** button.
2. Locate the **Proxy Server** area, select the "Use a proxy server" checkbox, and provide the following:
 - a. In the **Address** box, enter **localhost**.
 - b. In the **Port** box, enter the proxy listening port number discussed in the introduction above.
 - c. Click **OK**.

Note: Once defined, the proxy remains in effect whether the Debugger Client is operating or not. For Internet Explorer only, however, you can enable a proxy maintenance feature (as described [below](#)³⁸³) that automatically specifies and enforces the proxy definition when the Client is started. When the Client is closed, the feature restores the IE settings to their state before the Client was started.

Bypassing the proxy

To bypass the proxy server for certain connections:

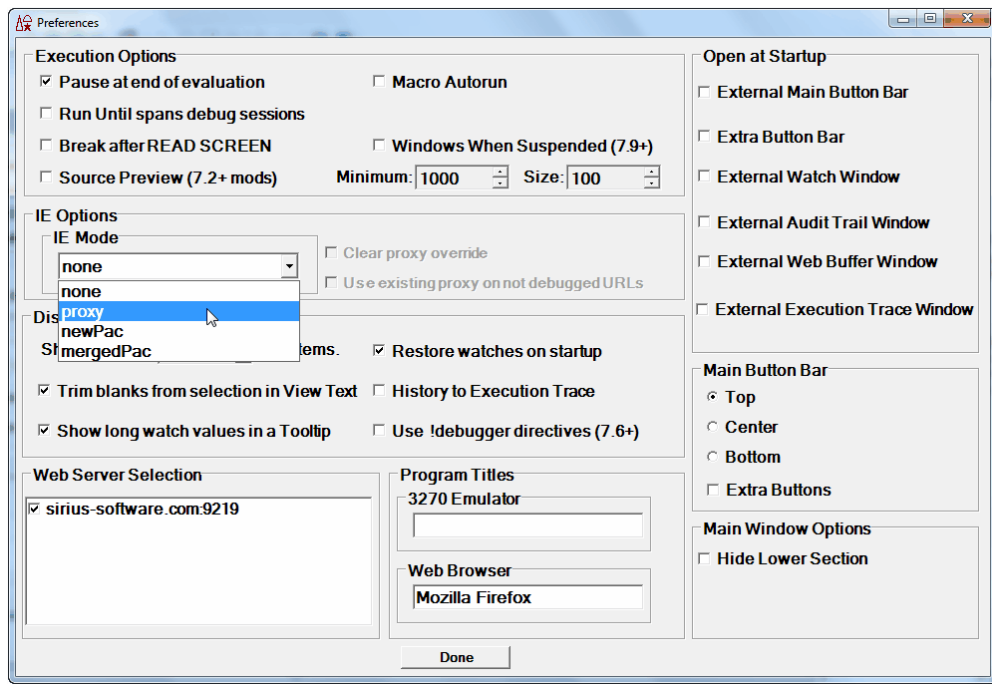
1. From the **Tools** menu, select **Internet Options**; then select the **Connections** tab, and click the **LAN Settings** button.
2. Locate the **Proxy Server** area, and click the **Advanced** button.
3. In the **Exceptions** area, in the list box labeled "Do not use proxy server for addresses beginning with," specify the URLs of any locations the Debugger Client should ignore.
4. Click **OK** as needed to close the multiple dialog boxes.

Automatically setting and removing the proxy definition

To invoke the proxy service automatically without having to manually define it each time (Internet Explorer only; for non-IE browsers, you must maintain proxy settings manually):

1. Select **Preferences** from the Debugger Client **File** menu to display the **Preferences** dialog box.
5. In the **IE Options** area, select the **proxy** option of **IE Mode**.

The initial **IE Mode** setting is **none** (browser settings are not maintained).



Note: The Client `setPreference`^[260] command also has an option that lets you switch the **IE Mode** option between **proxy** and **none**.

2. Click the **Done** button.

Now, whenever you start the Client, the Internet Explorer proxy settings are modified automatically, for all running browser instances.

The proxy maintenance feature remains on until you clear the checkbox (it persists between runs of the Client). When you shut down the Client or turn this feature off, the proxy modifications are removed, again for all instances of IE.

Only the following Internet Explorer settings are modified, and then restored:

- **Use automatic configuration script** (if specified) and **Address** values.
- **Use a proxy server** ... and its address and port.

These settings are found via the Internet Explorer **Local Area network (LAN) Settings** dialog box (accessed by **Tools** menu > **Internet Options** option> **Connections** tab > **LAN Settings** button).

The next subsection describes how to also include in the automatic maintenance any exception URLs specified to be bypassed by the proxy server.

Handling addresses specified to bypass the proxy server

It is an option of the automatic proxy maintenance to clear and save *at Client opening* any settings in the "Do not use proxy server for addresses beginning with:" box (accessed by **Tools** > **Internet Options** > **Connections** > **LAN Settings** > **Advanced**). The settings for these exceptions are then restored when the Debugger Client is closed (or when the feature is disabled).

To enable this feature, you must select *both* the **IE Mode** option **proxy** and the **Clear IE proxy override** checkbox in the Debugger Client's **Preferences** dialog box. The feature is off by default.

Note: If you do enable this feature, then exception URLs you had specified in the IE "Do not use proxy server for addresses beginning with:" box will now be routed to and passed through the Debugger when the Client is running. If the feature is not enabled (**Clear IE proxy override** is not selected), the operating system handles these exception URLs and the Debugger never sees them.

In the not very likely event that you want both to enable this feature and to have the exception URLs processed by your original proxy server, you can select the **Use existing IE proxy for URLs not to be debugged** checkbox in the **Preferences** dialog box (*in addition to* the **IE Mode** option **proxy** and the **Clear IE proxy override** checkbox).

In this case, the Debugger will process the exception URLs and pass them to the original proxy server. In fact, this option is designed primarily for troubleshooting purposes, as the Debugger Client **Audit Trail** page will then contain information about the disposition of these URLs.

Using the Client as proxy exclusively for designated addresses

You can use the Debugger Client as a proxy server *only* for requests for the hosts (Onlines) [specified](#)^[374] in the Debugger configuration file, **debuggerConfig.xml**. The IE browser is directed to do this host filtering by a script you create (by Client command or UI checkbox). Once this feature is in effect, the Client will act as a proxy server for requests for hosts specified in the script but for no other browser requests.

The Javascript script (**debuggerPac.js**) is called a Proxy Auto Config (PAC) file, and it is created by default in the Client [work-file folder](#)^[296]. If no such work folder is configured, the Client installation folder is used. The file location is specified in the IE **Tools** (in the **Address** value in **Tools** > **Internet options** > **Connections** > **LAN settings** > **Use automatic configuration script**). The Debugger Client will automatically create this file at Client startup, specify it in IE for you, and remove the specification from IE at Client shutdown.

Note: Caution is recommended when working with PAC files. Errors are not well reported and have the potential to prevent the browser from operating. Be very careful if you edit or create your own PAC files, and be sure to consult the Guide at: <http://www.proxypacfiles.com/proxypac/>

To invoke the automatic creation and maintenance of a PAC file:

- From the Client GUI:
 1. Select **Preferences** from the **File** menu, and locate the **IE Options** area.
 2. In the **IE Mode** box, select either **newPac** or **mergedPac**:
 - **newPac** generates a PAC file from **debuggerConfig.xml** and modifies the IE configuration settings to use it. Any existing designated PAC file is ignored.
 - **mergedPac** generates a PAC file like **newPac** except that it merges the freshly generated PAC file with any existing PAC file.
 3. Click the **Done** button.

No restart of IE or the Client is necessary. IE will run the script for each URL that is invoked from the browser.

- By Client command:

Issue the `setIEmode[258]` command. Use the **newPac** parameter to ignore any existing PAC file, or use the **mergedPac** parameter to merge any existing PAC file with the PAC file freshly generated from the settings in the **debuggerConfig.xml** file.

Lynx

The Lynx character-mode browser is fast and handy for testing. To make it use a proxy, set the environment variable **http_proxy** to the full URL of the Janus proxy.

Here is a Unix shell script example (running on a PC under Cygwin) that sets this variable and runs Lynx:

```
#!/bin/bash
http_proxy=http://127.0.0.1:8081
export http_proxy
env
lynx
```

Note: The **http://** is required.

Opera

To set up the proxy:

1. From the main Menu, select **Settings > Preferences > Advanced > Network > Proxy Servers**.
2. Select the **HTTP** checkbox, and enter **localhost** and the proxy listening port number discussed in the introduction above.

9.3.5 Test the end-to-end configuration

Subsections follow for the *Janus Debugger* and for the *Sirius Debugger*. It is assumed that the Debugger Client is up and running.

Janus Debugger

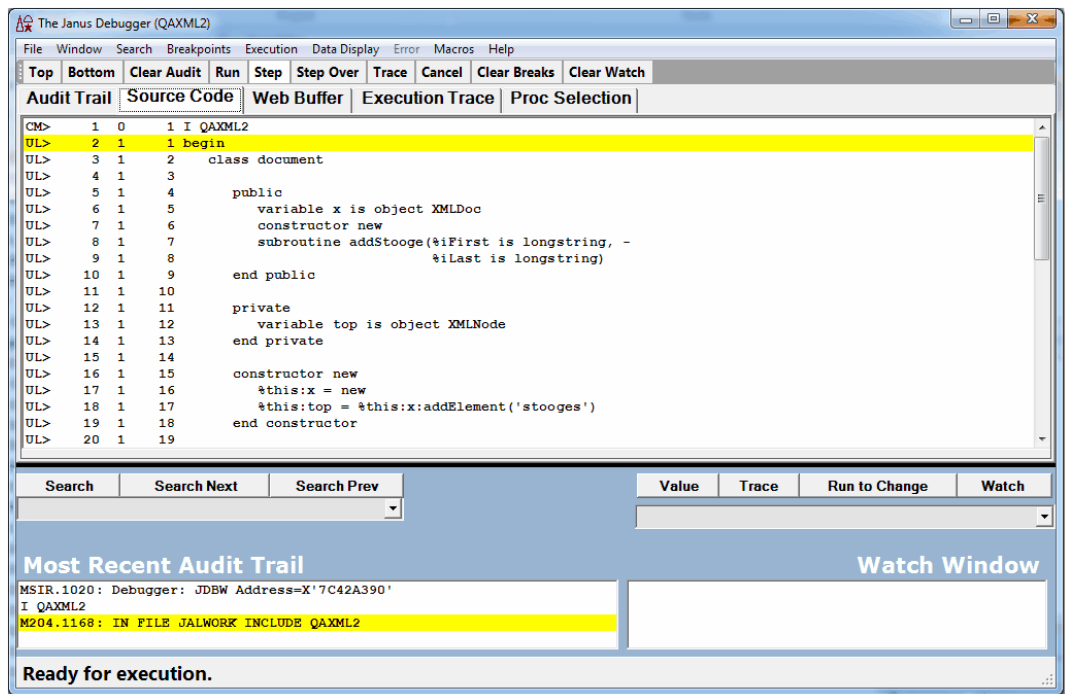
1. Open the browser you just [set up](#)^[380] to use the Debugger as a proxy.

Note: It is not uncommon at this point for the Client to display a benign [communication error message](#)^[354] when a browser connection to an external site gets closed. If this occurs, just click the **OK** button on the message box.

2. Invoke the URL of a User Language-based page from your target Janus Web Server.

Note: If you normally use "https://" and not "http://" to begin the URL (that is, your Web Server is SSL-secured), and if you [set up](#)^[374] the debuggerConfig.xml file for SSL support, make sure that you use "**http**://" here now to access the secured web port.

The source code of the program should appear in the **Source Code** page of the Client with its procedure name displayed in the title bar, similar to the following **Source Code** display:



If instead, you receive a Communication Error message that reports an "error while communicating with the remote host," you may have an error in the debuggerConfig.xml [settings](#)^[374]. If so, and you find the error, restart the Debugger Client and try the test URL again. For more information about error handling, see [How the Janus Debugger handles communication breaks](#)^[354].

The Debugger Client is ready to use. From the Client GUI, you can control the execution of your web application's User Language code (see [Getting Started](#)^[71]).

For an archive of information about features that are new or enhanced in the latest version of the Debugger Client, see the [Release Notes](#)^[395].

Sirius Debugger

1. From the *Model 204* command prompt or within a BATCH2 input stream, start a *Sirius Debugger* session:

```
SIRIUS DEBUG ON janClientPort pcHost pcPort workerPort
```

where:

janClientPort The name of the Janus client socket port that [is defined](#)^[368] for the *Sirius Debugger* to use to contact the Debugger Client workstation.

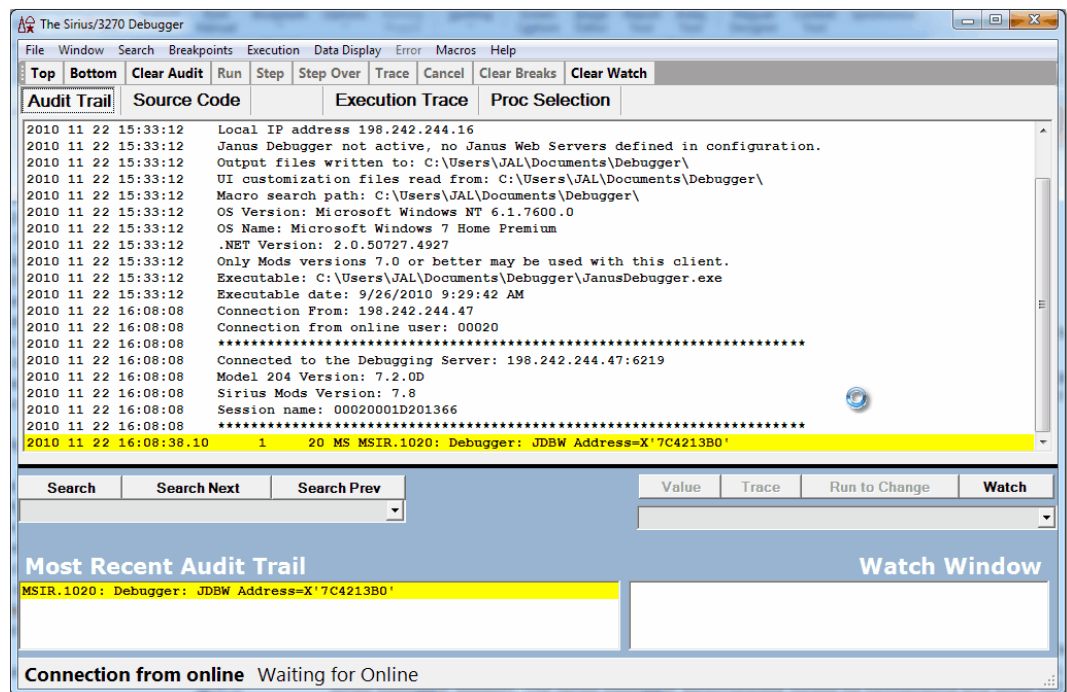
This port must be started.

- pcHost* The workstation running the Debugger Client. This may be an IP number or a DNS name, as [described earlier](#)^[370].
- pcPort* The workstation port number on which the Debugger Client is listening. As [described earlier](#)^[370], this is typically 8081.
- workerPort* The port number in your Online that [is defined](#)^[367] for worker threads. This can be the same port number that provides worker threads for the *Janus Debugger*, as well.

For example:

SIRIUS DEBUG ON DEBCLIENT 198.242.244.234 8081 3226

- Verify that you receive a message similar to:
***** MSIR.0915: Debugging is on; client is 198.242.244.234 port 8081, sessionID: 00000069D812279**
- On the Debugger Client, verify that a "busy" cursor displays, as well as a **Waiting for Online** message in the [Status bar](#)^[50]:



Additional **Audit Trail** page messages identify the Debugger Server's port and the address of its Online host, the user number assigned to the logged-in *Model 204* user, as well as the *Model 204* and *Sirius Mods* versions.

From this point on, any User Language program you initiate from the *Model 204* command line will appear in the **Source Code** tab of the Debugger Client GUI for debugging, the procedure name will appear in the Client's title bar, and the "busy" cursor and **Waiting for Online** message will display between requests until your session ends.

4. Turn off the *Sirius Debugger* by doing either of the following:
 - From the *Model 204* command prompt (or at the end of your BATCH2 stream), issue:

SIRIUS DEBUG OFF

You should receive this response in *Model 204*:

***** MSIR.0913: Sirius Debugger is now off**

On the Debugger Client, **Online has disconnected** displays in the Status area.

- Log off of *Model 204* (any logoff is an implied SIRIUS DEBUG OFF).

Note: Explicitly turning off the Debugger is necessary if you are using the *Janus Debugger* as well as the *Sirius Debugger* for the same Online and worker port. To switch from a *Sirius Debugger* session to a *Janus Debugger* session, you must explicitly drop the *Sirius Debugger* session. The *Janus Debugger* automatically closes its connections and does *not* require an explicit notification to switch or end a session.

5. Reissue the command from step 1 to restart the Debugger Client, and the Debugger Client is ready to use.

From the Client GUI, you can control the execution of your *Model 204* application's User Language code (see [Getting started](#)^[7]).

For an archive of information about features that are new or enhanced in the latest version of the Debugger Client, see the [Release Notes](#)^[395].

9.4 Providing updated versions of the Debugger Client

You can configure the Debugger to obtain updated builds of the Client from a central address, say, the URL of a local server. Then a Client menu item invokes a program (**updateGet.exe**) from which you download a new executable file (**JanusDebugger.exe**) to replace your existing Client.

This update maintenance feature assumes that each Client user has completed the initial Client installation, then uses this tool to refresh the executable as needed for fixes and enhancements. Currently, the Client does no checking of the version of the updated executable file, so you are responsible for determining when it is appropriate to download a new copy.

The updated executable file may be packaged in a zip file. If the file in the central URL has a .zip extension, the JanusDebugger.exe file is automatically extracted from it (an `unzip.exe` program is distributed).

To set up the feature:

1. Determine the URL where you will store updated Client executable files and prepare the mechanism by which the files are to be served from this URL.

The updating program sends an HTTP GET call for the JanusDebugger.exe or .zip file to this URL. You must provide code at this URL to respond to the GET.

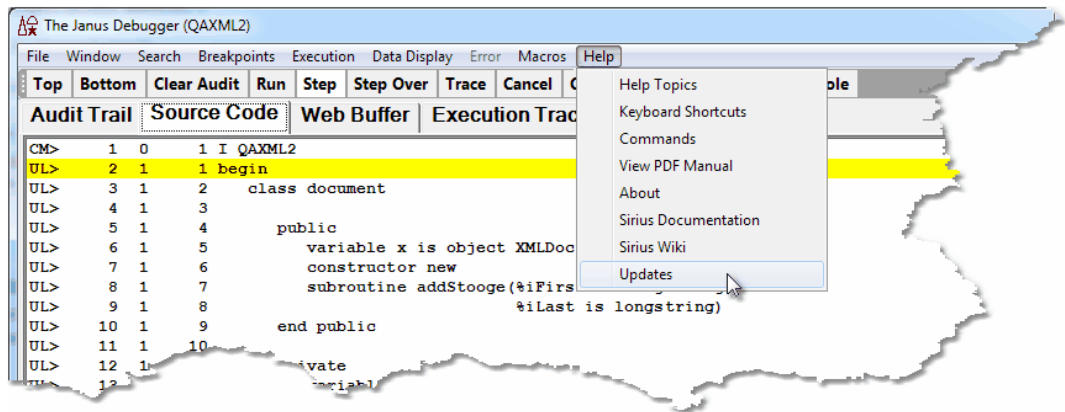
2. In the Debugger [Client configuration file](#)^[374] (`debuggerConfig.xml`), insert an `<updateURL>` element as a child of the `<debuggerConfig>` element.

This element must contain the URL (which must *not* be for an SSL port) at which you make available the updated Client executable file (and you may specify a .zip file).

When complete, your configuration file should have a structure like the following:

```
<debuggerConfig version="1.0">
  <serverList>
    .
    .
    .
  </serverList>
  <proxy>
    .
    .
    .
  </proxy>
  <updateURL>http://sirius-software.com:3224/janusdebugger.exe</updateURL>
  .
  .
  .
</debuggerConfig>
```

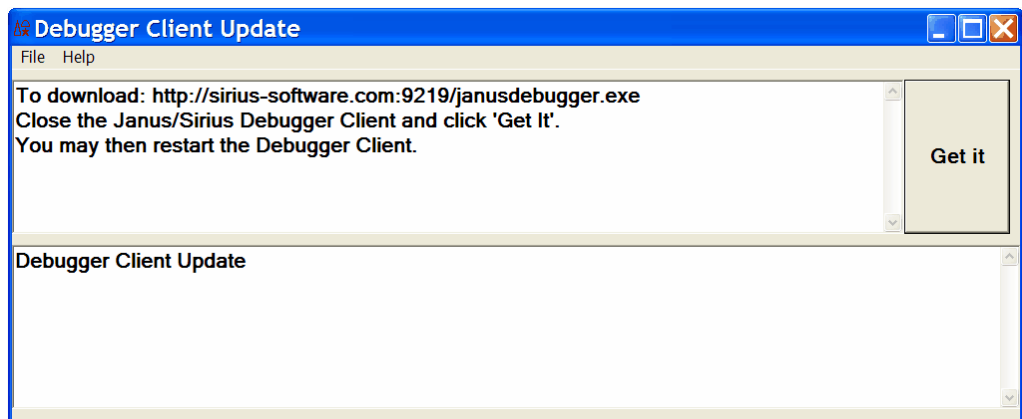
3. Check that the Debugger Client **Help** menu now contains the **Updates** option:



To get an updated version of the Client:

1. In the Client **Help** menu, click **Updates**.

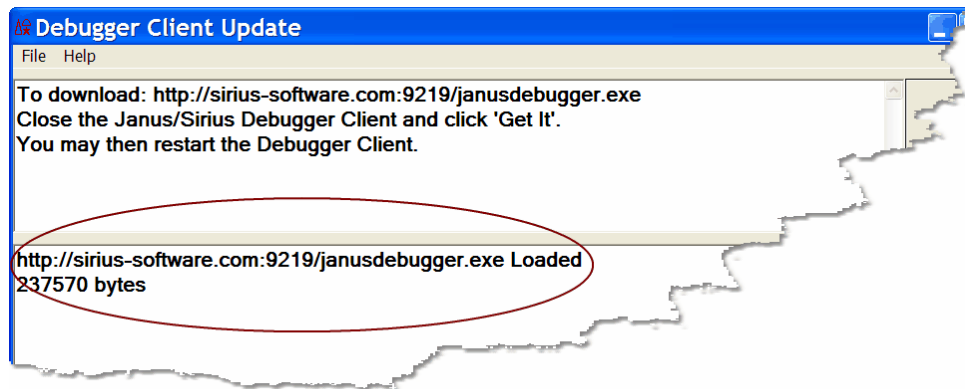
The **Debugger Client Update** dialog box (updateGet.exe program) is invoked, and it is passed the URL you specified in the `<updateURL>` tag in `debuggerConfig.xml`:



2. Close the Client; then press the **Get It** button.

The current JanusDebugger.exe is replaced with the new one. Closing the Client first avoids a file locking error.

You should see a confirmation message like the following:



CHAPTER 10 *Release Notes*

This section contains brief descriptions of the features added in each update, or "build," of the Debuggers since their first commercial release. You can determine the current build number of the Debugger Client by selecting the **About** option of the Client **Help** menu.

Sirius Mods Versions 7.0 - 8.0

Build: 62 (31 March 2013)

Note: Build 53 or higher of the Client is recommended for sites running under version 7.8 or higher of the *Sirius Mods*.

These are the principal changes to the Debugger Client since the previous build:

Physical changes and additions to the Client user interface:

- New [Preferences](#)^[18] window option:
The new **IE Options** area and **IE Mode** drop-down menu replace the former **Web Browser Options** area and **Automatically Maintain IE proxy settings** option. In addition, **IE Mode** options are available to invoke automatic creation and maintenance of [Proxy Auto Configure \(PAC\) files](#)^[385], which designate the only host URLs the Debugger Client will handle requests for as a proxy server.
- The **Proc Selection** page now has [Black List filtering](#)^[79] to accompany the already existing White List filtering, and your filtering selection now persists over runs of the Client.

Macro and command changes:

New commands:

- [disableButton](#)^[202] and [enableButton](#)^[204] commands disable and enable Client button bar buttons, and [labelButton](#)^[219] lets you replace the label of a button.
- [generatePac](#)^[209] command restricts the Debugger Client's role as-proxy-server to handling only Internet Explorer requests for hosts (onlines) specified in `debuggerConfig.xml`.
- [setBlackList](#)^[256] and [setWhiteList](#)^[265] commands let you temporarily override (but not physically affect) the contents of an existing procedure Black List or White List file.

- [setIEMode](#)^[258] controls whether, and the host URLs for which, the Debugger will serve as the proxy server for users of the Internet Explorer browser.
- [setTitle](#)^[264] and [restoreTitle](#)^[243] let you change the title of the Client's main window.
- [showIE](#)^[267] displays the current IE browser operating mode (the [setIEMode](#)^[258] parameter setting that is in effect), and displays the current values of IE settings that pertain to the Debugger Client.
- [turnOnBlacklist](#)^[277], [turnOffBlacklist](#)^[276], and [reloadBlacklist](#)^[238] are analogous to the already-existing [turnOnWhitelist](#), [turnOffWhitelist](#), [reloadWhitelist](#) commands.

Changed commands:

- The [continueIf](#)^[197] and [continueMacroIf](#)^[198] macro-only commands now support expressions.

New Client functions:

- [&¤tTitle](#)^[330] and [&&originalTitle](#)^[337] display the current and default titles of the Client main window.
- [&&exists](#)^[331] tests if a macro variable is defined.
- [&&blackOrWhiteList](#)^[326] tests whether black-list filtering, white-list filtering, or neither is in effect.
- [&&ieMode](#)^[333] returns the setting of the [IE Mode](#)^[19] option in the Client's **Preferences** window.

Other changes:

- Documentation added for [setting up a Chrome browser as a Client proxy](#).^[381]

Build: 61 (31 November 2012)

- ▣ These are the principal changes to the Debugger Client since the previous build:

Macro and command changes:

- [nsLookup](#)^[230] command added to aid in debugging setup problems
- Assorted bug fixes.

Build: 60 (31 October 2012)

- ▣ These are the principal changes to the Debugger Client since the previous build:

Physical changes and additions to the Client user interface:

- Status message improvements:
 - Added a display of the value (if ten characters or less) to the message received after a successful "[run until variable = value](#)"^[134] execution.
 - The [macroTrace](#), [mapButton](#), [mapKey](#), [clearButton](#), and [setPreference](#) commands now report after a successful operation (in the [Status bar](#)^[50], [Console](#)^[316], and Client log).

Macro and command changes:

- New [help](#) command options:
 - [preference](#)^[212] displays the preferences available to users via options of the [setPreference](#)^[260] command.
 - [ignoredFiles](#)^[212] displays the Debugger's [filtered file types](#)^[376].
- New [ignoredFileTypeList](#)^[260] option of the [setPreference](#) command allows on-off toggling of file-type filtering.

New Client functions:

- [&&exists](#)^[331] tests if a macro variable is defined.
- [&¤tRunningMacro](#)^[328] returns the file-system path of a currently running macro.
- [&&preference](#)^[337] returns the current value of a specified Client preference.

Build: 59 (17 August 2012)

- ☐ These are the principal changes to the Debugger Client since the previous build:

Physical changes and additions to the Client user interface:

- Help menu [Commands](#)^[38] option now displays individual command syntax
- [Command line](#)^[317] tool now maintains persistent history of commands, reviewable via the keyboard up/down arrow keys
- [macroTrace](#)^[222] processing log now shows syntax help if commands have syntax errors
- [Watch Window color panel](#)^[296] option (**Out of Scope**) for **Watch Window** items not in code in currently active tab (**Source Code** or **Daemon**).
- Degree of transparency of Preferences and [external-button](#)^[42] windows is now [user-settable](#)^[307]

- Search box widened for ["Hide Lower" mode](#)^[305]
- Selecting **Add Watch** (from context menu for a **Source Code** line) automatically opens external **Watch Window** if Client in ["Hide Lower" mode](#)^[305]
- Support master thread and daemon thread [interaction](#)^[143] ([ReturnToDaemon](#) method/[Continue](#) method)

At least version 8.1 of the *Sirius Mods* is required.

Macro and command changes:

- New commands:
 - [setStatusMessage](#)^[262] lets you set a message in the Client [Status bar](#)^[50]
 - [unSet](#)^[279] deletes [macro variables](#)^[319]
- `<>` ("not equal" operator) added to [assert](#)^[180] command
- Argument added to [help](#)^[212] command to display help for a specific command or function
- New Client functions:
 - [&&statusMessage](#)^[340] returns most recent Client [Status bar](#)^[50] message
 - [&&numberOfLevels](#)^[336] returns the number of source levels being debugged
 - [&&amDaemon](#)^[322] returns 1 if debugging a daemon, or 0 if not
 - [&&selectedTab](#)^[340] returns label of selected main window tab, or "" if none

Build: 58 (30 April 2012)

- ☐ These are the principal changes to the Debugger Client since the previous build:

Physical changes and additions to the Client user interface:

- New **Help** menu option (**Functions**) to display [Client functions](#)^[321]
- Search and **Watch Window** controls display at bottom of main window when in ["Hide Lower" mode](#)^[305]

Macro and command changes:

- [Macro variables](#)^[319] now allowed as arguments to Client functions
- New client functions:
 - [&&getMainSearchInputArea](#)^[332]
 - [&&getVariableOrFieldInputArea](#)^[332]
 - [&&isWatched](#)^[334]

- [&&numberWatched](#)^[336]
- [&&numberOfBreakpoints](#)^[335]
- [&&sum](#)^[341]
- [&&windowStatus](#)^[343]
- String functions:
 - [&&concatenate](#)^[327]
 - [&&index](#)^[334]
 - [&&length](#)^[335]
 - [&&substring](#)^[341]
 - [&&verifyMatch](#)^[342]
 - [&&verifyNoMatch](#)^[343]
- New commands:
 - [clearStatus](#)^[192] clears current messages from Client [Status bar](#)^[50]
 - [feoDisplay](#)^[208] displays current occurrence value in an FEO loop
 - [kill](#)^[218] stops a running macro (like Kill Running Macro option of **Macros** menu)
 - [reloadLists](#)^[239] reloads the [Exclude/Include proc/routine lists](#)^[70]
 - [removeCurrentWatch](#)^[240] removes currently selected **Watch Window** item
 - [showFunctions](#)^[267] displays all the Client functions in alphabetical order
- Command argument of [loadWatch](#)^[220] and [saveWatch](#)^[246] allows absolute path name
- New option ([valueDisplayOnConsole](#)) of [setPreference](#)^[260] command to control whether [value displays](#)^[101] appear in a separate **Value** window when the **Console** window is open
- [Backslash \(\\) escape character](#)^[45] for searches for strings beginning with an ampersand (&)
- [&&arg](#)^[323] function now restricted to commands *within* macros (in previous builds, [&&arg](#) could be used outside of a macro, but its results were unpredictable and unreliable)
- Client functions and constants now allowed on [continueIf](#)^[197] and [continueMacroIf](#)^[198]

Build: 57 (31 January 2011)

- ☐ These are the principal changes to the Debugger Client since the previous build:

Physical changes and additions to the Client user interface:

- New [Preferences](#)^[18] window options:
 - **Main Button Bar** options (**Top**, **Center**, **Bottom**) allow [changing the position](#)^[41] of the main (non-external) button bar
 - **Main Button Bar** option (**Extra Buttons**^[21]) to [add extra buttons to the main button bar](#)^[44] instead of to a second button bar
 - **Main Window Options** option (**Hide Lower Section**^[21]) to [hide or restore the lower section of the Client main window](#)^[305]
 - More **Open at Startup** options to [automatically open external windows](#)^[21] at Client startup (for the **Watch Window** and **Audit Trail**, **Web Buffer**, and **Execution Trace** pages)
- A [Step out](#)^[64], [Step over](#)^[56], or [trace](#)^[130] followed by pressing the Enter key launches another step out, step over, or trace if program execution has not completed
- Named buttons may be replaced by [separators](#)^[285] for visual separation between buttons
- [Edit uimore.xml](#)^[22] option added to **File** menu
- Pinned external windows:
 - Context-menu options simplify [pinning](#)^[303] and unpinning
 - [Pin icon](#)^[303] in title bar
- Additional StringTokenizer class variables are [viewable](#)^[113]: **CurrentQuoted**, **CurrentToken**, **String**, and **StringLength**

At least version 8.0 of the *Sirius Mods* is required.

Macro and command changes:

- New "global" versions of existing Client functions:
[&&globalAssertFailureCount](#)^[332], [&&globalAssertSuccessCount](#)^[333], and [&&globalAssertStatus](#)^[333]
- New commands:
 - [clearButton](#)^[189] removes all mappings from a Client named button
 - [clearKey](#)^[191] removes all mappings from a Client hot key
 - [evaluate](#)^[205] builds and runs a Client command

- [hideLower](#)^[213], [restoreLower](#)^[243], and [toggleLower](#)^[273] commands for hiding and restoring the lower section of the Client main window
- [increment](#)^[215] and [decrement](#)^[201] add or subtract 1 from a numeric-valued macro variable
- [mainButtonBar](#)^[223] positions the main button bar within the Client window
- [mapKey](#)^[226] command assigns keyboard shortcuts without requiring a mapping file or Client restart
- [resetGlobalAssertCounts](#)^[241] clears counts of new "global" Client functions
- [setBreakpointOnCurrentLine](#)^[257] and [clearBreakpointOnCurrentLine](#)^[188] expand flexibility of the commands for controlling breakpoints
- Macro-only commands:
 - [continueMacroIf](#)^[198] conditionally processes the macro that contains it
 - [includeIf](#)^[214] conditionally calls another macro
 - [macroWait](#)^[223] slows down macro execution
- New option (**main**) of [extraButtonBar](#)^[207] command to combine the main and second button bars
- New option (**ieAuto**) of [setPreference](#)^[260] command to control automatic proxy maintenance for the Internet Explorer browser
- "[In window] command" format and command parameters now allowed in [command mappings](#)^[288] for Client button, keyboard shortcuts, and macros
- Button-mapping keyword (**separator**) for [mapButton](#) command or mapping-file **mapping** element converts a button to a [visual separator](#)^[285]

Build: 56 (22 August 2011)

- ☐ These are the principal changes to the Debugger Client since the previous build:

Physical changes and additions to the Client user interface:

- Optional [second button bar](#)^[43]
- Former **Window** menu option **Open External Button Window** renamed to [Show Main Button Bar in External Window](#)^[26]
- New **Preferences** window section [Open at Startup](#)^[21] with checkboxes for button bars:
 - **External main button bar**
 - **Extra button bar**

- On external windows, [Enter key searches down](#)^[301] when the search text box has focus

Macro and command changes:

- New commands control the [pinning of external-windows](#)^[303]: **pin** and **unPin**
- New [extraButtonBar](#)^[207] command launches an extra button bar
- Addition to **buttonBar** command: new option [show](#)^[186]
- New [mapButton](#)^[224] command assigns button mappings without requiring a mapping file or Client restart
- Other new commands:
 - [resetAssertCounts](#)^[241] clears counts of Client functions
 - [restart](#)^[242] and [restartDefault](#)^[242] restart the Client
- The descriptions of the "macro-only" commands (**continueIf**, **include**, **macroTrace**, **noSpan**, **setM204Data**, and **span**) are moved to the [Client command reference](#)^[179]

Build: 55 (15 May 2011)

- ▣ These are the principal changes to the Debugger Client since the previous build:

Physical changes and additions to the Client user interface:

- **Dock** and **File** menus added to the [external Button Bar](#)^[42]
- **External Button Bar** now [dockable](#)^[42] (via **Dock** menu or [buttonbar](#)^[186] command)
- [Search buttons](#)^[44] added to bottom of work windows and external windows
- Three additional [Client buttons](#)^[285] (bringing total to fifteen) for mapping Client commands
- New **Proc Selection** tab [checkboxes](#)^[70] turn on or off the inspection of lists of procedures/routines to exclude or include in source code displays in the Client

Macro and command changes:

- "[In window] command" format applies the searching commands to most any Client window (main, external, or work), as follows:

In window	command
Is one of:	Is one of:
about ^[39]	bottom ^[183]
auditTrail ^[10]	searchDown ^[247]
commands ^[38]	searchFromBottom ^[248]
console ^[316]	searchFromTop ^[250]
executionHistory ^[134]	searchUp ^[251]
executionTrace ^[13]	top ^[274]
keyboardShortcuts ^[38]	
source ^[11]	
textViewer ^[149]	
value ^[101]	
watchWindow ^[15]	
webBuffer ^[12]	

- [windowToTop](#)^[281] now supports trailing wildcard searches
- [getVariablesForClass](#)^[211] display window now is non-modal (user interaction not required)
- New [buttonbar](#)^[186] command
- [setPreference](#)^[260] command added for turning on and off the various code exclude/include options

Other changes:

- New parameter values and defaults for [SIRIUS DEBUG ON](#)^[152] command
At least version 7.9 of the *Sirius Mods* is required.
- Procedure name searches may now use [asterisk and question mark wildcards](#)^[76] in any position, as well as double quotation marks to escape them
At least version 7.9 of the *Sirius Mods* is required.
- Number of code blocks you can [exclude/include](#)^[68] from or in debugging in a single request increased to 300
At least version 7.9 of the *Sirius Mods* is required.

Build: 54 (28 February 2011)

☐ These are the principal changes to the Debugger Client since the previous build:

- Statement execution history now [viewable in Source Code \(or Daemon\) tab](#)^[136],

controlled by:

- New Execution menu options: [Select Previous History Line](#)^[30], [Select Next History Line](#)^[30], [Select First History Line](#)^[30], [Select Last History Line](#)^[31]
- New Client commands: [previousHistory](#)^[237], [nextHistory](#)^[228], [firstHistory](#)^[208], [lastHistory](#)^[220]
- [Requirement](#)^[290] to have button mappings for the Client **step**, **run**, and **cancel** commands is removed
- Macro and command changes:
 - New program-window position commands: [moveBrowserToTop](#)^[227] and [moveTn3270ToTop](#)^[227]
 - The "macro-only" restriction for the **set** and **assert** commands is removed, and their descriptions are moved to the [Client command reference](#)^[179]
 - New Client commands:
 - [previousHistory](#)^[237], [nextHistory](#)^[228], [firstHistory](#)^[208], [lastHistory](#)^[220]
 - [openExternalButtonWindow](#)^[232], [closeExternalButtonWindow](#)^[194]
- Physical changes and additions to the Client user interface:
 - Button bar may be [moved to an external window](#)^[42] via **Open External Button Window** option of **Window** menu
 - New [Search Prev button](#)^[44] searches backwards in the current Client tab (same as Alt+**Search** button)
 - New **Windows When Suspended** option in **Preferences** dialog box [brings up 3270 emulator window](#)^[63] or [browser window](#)^[144] for code not actively being debugged
 - **Get History** option of [Execution menu](#)^[29] renamed to **Get/Display History**
 - [Preferences](#)^[18] dialog box is no longer modal: it can remain open while the Client is active
 - [Command Line](#)^[317] window now remains on top of your PC's window stack

Build: 53 (30 November 2010)

☐ These are the principal changes to the Debugger Client since the previous build:

- Build 53 or higher of the Client is recommended for sites running under version 7.8 of the *Sirius Mods*.
- Debugger Client now insists on Version 7.0 or greater of the *Sirius Mods* (attempts to use earlier *Sirius Mods* versions produce an error message)

- [Browsing-while-debugging](#)^[381] now allowed for browser on which the Client is defined as proxy server
- Physical changes and additions to the Client user interface:
 - [Button bar](#)^[39] moved above main window, and [Status area](#)^[50] moved to strip at bottom of main window
 - Font size in most Client windows is now [scalable](#)^[299]
 - Two additional [Client buttons](#)^[285] (bringing the total to twelve) now available for mapping Client commands
 - New [Help menu](#)^[38] documentation links, whose default destinations are also [configurable](#)^[379]:
 - **Sirius Documentation** (links to Documentation page of the Sirius website)
 - **Sirius Wiki** (links to Sirius wiki for *Model 204* information)
 - Editor for Client text-editing tasks can now be [set in debuggerConfig.xml file](#)^[379]
 - [READ SCREEN information](#)^[62] now displayed in the Client's **Audit Trail** tab
 - Mouse double-click now opens [external windows](#)^[300]
 - Removed label **Variable or Field** for text box above **Watch Window**; changed **Trace Changes** button label to **Trace**; changed the former **Trace** button label to **Trace All**; changed the former **Add Watch** button label to **Watch**.
- Macro and command changes:
 - [Macro command line](#)^[317] tool is renamed to **Command Line** and now accepts and runs non-macro commands
 - New macro-related commands:
 - [openCommandLine](#)^[231] and [closeCommandLine](#)^[193] toggle the **Command Line** tool accessed from the **Macros** menu
 - [openMacroConsole](#)^[234] is now a synonym for the **macroConsole** macro command, which opens the **Macro Console**; and [closeMacroConsole](#)^[196] closes the console
 - [createMacro](#)^[200] mimics the action of the **Macros** menu **New Macro** option
 - [editMacroFromUISelection](#)^[204] mimics the action of the **Macros** menu **Edit Macro** option
 - [runMacroFromUISelection](#)^[244] mimics the action of the **Macros** menu **Run Macro** option
 - New command and method options for sending user interface commands to the Client:

- [CLIENTCOMMAND](#)^[155] subcommand of SIRIUS DEBUG
- DebuggerTools class additions:
 - New [ClientCommand](#)^[164] method
 - New [StatusMessage](#)^[166] method
 - New parameter (to execute Client command) for [Break](#)^[162] method
 - New parameter (to execute Client command) for [Command](#)^[164] method
- The "macro-only" restriction for some commands ([echo](#), [toggle](#), [varDump](#), and [windowToTop](#)) is removed, and their descriptions are moved to the [Client command reference](#)^[179]

Build: 52 (9 August 2010)

▣ These are the principal changes to the Debugger Client since the previous build:

- Bug fixes only.

Build: 51 (15 July 2010)

▣ These are the principal changes to the Debugger Client since the previous build:

- New Preferences (File > Preferences) option (**Use existing IE proxy for URLs not to be debugged**) for troubleshooting [IE proxy server](#)^[385] issues
- Some [mapping errors](#)^[290] in the [ui.xml](#) and [uimore.xml](#) files will now invalidate only the individual item and not the entire file
- New [selectWatchWindow](#)^[254] command to give focus to an external **Watch Window**
- Debugger Client now targets [.NET Framework 3.5 SP1](#)^[369]

Build: 50 (31 March 2010)

▣ These are the principal changes to the Debugger Client since the previous build:

- **Audit Trail**, **Web Buffer** and **Execution Trace** windows may now be moved to separate [external windows](#)^[300]
- Client [Window menu](#)^[24] lists and tracks new Client external windows
- Multiple issues fixed concerning Client operation on a Windows 7 host, thereby making build 50 the preferred Client build for Windows 7 users

- Client's display and watch facilities now work for *Janus* SOAP Arraylist objects the same as they do for [\\$lists or Stringlists](#)^[104]
- New **Serialize** option in context menu for XmlDocument or XmlNode object variables in the **Watch Window** for [displaying serialized object content](#)^[106]
- New Preferences (**File > Preferences**) option (**Macro Autorun**) [automatically runs a macro](#)^[318] when a same-named procedure is debugged
- New macro-only commands:
 - [macroTrace](#)^[222] traces all macro statements in the debugging session
 - [setM204Data](#)^[260] sets a *Model 204* %variable or global variable
 - [windowToTop](#)^[281] brings specified window to top of window stack
- New [Client functions](#)^[321] (&&assertFailureCount, &&assertSuccessCount, &&assertStatus) report macro **assert** command results
- New mappable commands for Client windows:
 - [clearHistory](#)^[190] and [closeHistory](#)^[195] affect the [Execution History window](#)^[134]
 - [clearMacroConsole](#)^[191] and [closeMacroConsole](#)^[196] affect the [Macro Console window](#)^[316]
 - [closeExternalAuditTrailWindow](#)^[193] closes an [external](#)^[300] **Audit Trail** window
 - [closeExternalExecutionTraceWindow](#)^[194] closes an external **Execution Trace** window
 - [closeExternalWatchWindow](#)^[194] closes an external **Watch Window**
 - [closeExternalWebBufferWindow](#)^[195] closes an external **Web Buffer** window
 - [closeExternalWindows](#)^[195] closes any open external windows
 - [closeValueDisplay](#)^[196] closes [Value windows](#)^[101]
 - [openExternalAuditTrailWindow](#)^[231] opens an [external](#)^[300] **Audit Trail** window
 - [openExternalExecutionTraceWindow](#)^[232] opens an external **Execution Trace** window
 - [openExternalWebBufferWindow](#)^[233] opens an external **Web Buffer** window
- New [showCommands](#)^[266] mappable command displays all the Client commands in alphabetical order
- Boolean enumeration variables (for example, %bool1 is enumeration Boolean) may now be [set](#)^[124] (only to True or False)

At least version 7.7 of the *Sirius Mods* is required.

- Single quotes now allowed on string literal values in macros, wherever string literals may appear (for example with [&&prompt](#)^[315])
- Search path for PDF copy of [product User's Guide](#)^[371] (**View PDF Manual** option, Client **Help** menu) now first tries the product installation folder, then the Doc subfolder
- PDF version of Debugger *User's Guide* is [installed](#)^[371] in the target installation folder instead of in a **Doc** subfolder.

Build: 49 (31 December 2009)

☐ These are the principal changes to the Debugger Client since the previous build:

- New macro commands: [loadWatch](#)^[220] and [saveWatch](#)^[246]
- Missing **Save** option added to **File** menu on [Value windows](#)^[101]
- New Client menu option (**External Watch Window**) and command ([openExternalWatchWindow](#)) [move the Watch Window to an external window](#)^[300]
- The [assert](#)^[180] command now accepts a Client function as a target (left side)

Build: 48 (31 October 2009)

☐ These are the principal changes to the Debugger Client since the previous build:

- Added **Proc Selection** tab buttons for creating and editing Exclude and Include lists for [selectively excluding from debugging](#)^[67] the code for particular methods and User Language subroutines
- New Debugger configuration file (debuggerConfig.xml) elements for [modifying the default location of Client work files](#)^[296]
- New [Client functions](#)^[321] (&&searchSuccess, &&searchResult, &&procName)
- [Binary view](#)^[102] of a variable's value now also shows printable characters
- Client [Value display windows](#)^[101] now are non-modal for ease of use

Build: 47 (30 September 2009)

☐ These are the principal changes to the Debugger Client since the previous build:

- New Client menu option (**Toggle Init Exclude**) and command (**toggleInitExclude**) control whether to invert the way [exclude mode](#)^[67] operates: initially excluding code instead of initially including code (until an explicit directive)

At least version 7.6 of the *Sirius Mods* is required.

- Added [Edit White List button](#)^[82] to **Proc Selection** tab
- Added exclude/include procedure lists when [debugger directives](#)^[67] are on

Build: 46 (15 September 2009)

☐ These are the principal changes to the Debugger Client since the previous build:

- Source Code lines can be explicitly [excluded from Debugging](#)^[67]
- Debugger commands and macros may now be [mapped](#)^[286] to numeric keys as well as to alphabetic and function keys
- Multiple [commands](#)^[285] added for Client tab selection and navigation:

- selectAuditTab
- selectSourceTab
- selectWebBufferTab
- selectExecutionTraceTab
- selectProcSelectionTab
- selectNextTab

- [pafgi command](#)^[234] added for viewing field groups

At least version 7.6 of the *Sirius Mods* is required.

- Options **PAI** and **PAFGI** added to [Data Display menu](#)^[33]
- Additional text items available for [user-specified coloring](#)^[293] in the Client windows:
 - User Language TEXT and HTML statement blocks
 - User Language Macro Facility statements
- **Add Watch** and **Display** right-click options for lines within HTML or TEXT blocks now detect *Model 204* [field-name expressions](#)^[96]
- New Client [Preferences option](#)^[18] (**Show long watch values in a Tool Tip**) for [viewing Watch Window items](#)^[90] too wide to fit

- [\\$CURREC](#)^[97] calls as well as [\\$FIELDGROUPID](#) and [\\$FIELDGROUPOCCURRENCE](#)^[121] calls can now be quickly evaluated/watched
At least version 7.6 of the *Sirius Mods* is required.
- Added the Value property to the [expansion](#)^[113] of a ScreenField object

Build: 45 (30 July 2009)

▣ These are the principal changes to the Debugger Client since the previous build:

- Additional class variables are [viewable](#):^[113]
 - Record class: RecordNumber and FileName variables
 - RecordsetCursor class: RecordNumber, FileName, LoopLockStrength, and State variables
 - FastUnloadTask class: State variable
 - StringTokenizer class: AtEnd and NotAtEnd variables
 - UserStatistics class: LoginToString, RequestToString, and ToString variables

At least version 7.6 of the *Sirius Mods* is required.

For all versions of the *Sirius Mods*, object variables belonging to these classes are recognized as system objects, but no variables are suitable for display:

- CharacterMap class
- RandomNumberGenerator class
- Support added for [viewing fields](#)^[95] that belong to Model 204 field groups
At least version 7.6 of the *Sirius Mods* is required.
- [pai](#) command added for [viewing all fields](#)^[117] in a record
At least version 7.6 of the *Sirius Mods* is required.

Build: 44 (09 July 2009)

▣ These are the principal changes to the Debugger Client since the previous build:

- Fixes for some cases where string data was erroneously shown in binary/hex notation (X'aabbcc' format)
- FloatNamedArraylist class variables now [viewable](#)^[111]

Build: 43 (30 June 2009)

☐ These are the principal changes to the Debugger Client since the previous build:

- [Global variables referenced by \\$GETG](#)^[96] calls can now be quickly evaluated/watched by right-clicking their program line and selecting **Add Watch** or **Display** from the context menu
- [\\$STATUS and \\$STATUSD](#)^[97] calls can now be quickly evaluated/watched by right-clicking their program line and selecting **Add Watch** or **Display** from the context menu

At least version 7.6 of the *Sirius Mods* is required.

- New **File** menu item ([Restart with Default Window Size](#)^[18]) restarts the Client and displays it with the size and position it occupied when it was first installed
Otherwise, the overall size and position of the Client window as well as the dimensions of its internal windows is stored when the Client closes and reused when the Client is next opened.
- New **Macros** menu items ([Run and Edit](#)^[36]) identify the most recently run macro and enable quick re-execution or editing of that macro
- New mappable command ([clearWebBuffer](#)^[193]) and **Window** menu item (**Clear Web Buffer**) that clear the contents of the **Web Buffer** tab.
- New mappable commands that help to debug requests that contain objects, lists, or Stringlists. Probably better suited for macros, the commands may also be mapped to buttons or keys:
 - [expandObject](#)^[206] displays a list of object variable names and values for its object instance argument
 - [expandList](#)^[205] displays the list items in the \$list or Stringlist referenced by the variable specified as the command argument
 - [getVariablesForClass](#)^[211] displays a list of the names of the variable members in the class specified as the command argument
- New mappable command ([valuedisplay](#)^[279]) that, like the [Value button](#)^[101], displays in a separate window the value of the item currently in the **Variable or Field** area
- New mappable command ([showShortcuts](#)^[268]) that displays the Client's current (default as well as mapped) keyboard shortcuts
- New macro-only command ([macroConsole](#)^[221]) that invokes the [Macro Console](#)^[316] window.
- Class variable members in system exception classes are now [viewable](#)^[111]
- Increased drop-down history of search terms, procedure names, and variables from ten to twenty
- Multiple bug fixes (listed in Client's **Help** menu, **About** option).

Build: 42 (27 March 2009)

☐ These are the principal changes to the Debugger Client since the previous build:

- [User-specified](#)^[293] colors for text and backgrounds in the Client windows
- Independent [coloring](#)^[295] for **Source Code** comments

At least version 7.6 of the *Sirius Mods* is required for comments bounded by characters defined by COMSTART and COMEND commands.

- Ability to view [dummy string substitutions](#)^[127] in *Model 204* commands (formerly only in User Language statements)

At least version 7.6 of the *Sirius Mods* is required.

Build: 41 (30 January 2009)

☐ These are the principal changes to the Debugger Client since the previous build:

- Additions to the **Preferences** dialog box:
 - **3270 Emulator** option added to [bring up 3270 emulator window](#)^[63] at pause for user input
 - **Web Browser** option added to [bring up browser window](#)^[144] at pause for user input
 - **Clear IE Proxy Override** option enhances [IE automatic proxy server maintenance](#)^[382]
- New [session highwater mark](#)^[156] item added to the output display of the SIRIUS DEBUG STATUS command.

At least version 7.5 of the *Sirius Mods* is required.
- New **File** menu option ([Edit debuggerConfig.xml](#)^[18]) opens the **debuggerConfig.xml** file for editing
- "Web request will NOT be debugged" [audit trail messages](#)^[376] now indicate reason
- Debugger Client now always becomes the topmost window when a [breakpoint](#)^[58] is reached
- The **Add Watch** button now works properly with %variables whose names also have an embedded per cent (%x) character or whose names are enclosed in curly braces {%x}.

Sirius Mods Versions 7.0 - 7.3

Build: 40 (8 December 2008)

☐ These are the principal changes since the previous build:

- `debugTrace` tag added to `debuggerConfig.xml` file for additional diagnostic Client logging

Build: 39 (15 October 2008)

☐ These are the principal changes since the previous build:

- Bug fixes only.

Build: 38 (30 September 2008)

☐ These are the principal changes since the previous build:

- The Dataset object is added to the list of [viewable](#)^[111] classes.
At least version 7.2 of the *Sirius Mods* is required.

Build: 37 (31 August 2008)

☐ These are the principal changes since the previous build:

- New **Clear Execution Trace** [menu item](#)^[24] and `clearExecutionTrace` command for clearing the contents of the **Execution Trace** tab
- Ability to display statement [execution history](#)^[134] in **Execution Trace** tab
- **Clear** button added to **Execution History** and **Macro Console** windows.
- New macro-related commands: [toggle](#)^[271], [continueif](#)^[197], [span](#)^[269], and [noSpan](#)^[229].
- New optional attribute, [startUpMacro](#)^[287], for the `mappings` element in a `ui.xml` file mapping of alternative Client buttons, hot keys, and macros

Build: 36 (31 July 2008)

☐ These are the principal changes since the previous build:

- Improved [display of statement execution history](#)^[134]

At least version 7.3 of the *Sirius Mods* is required.

Build: 35 (11 July 2008)

▣ These are the principal changes since the previous build:

- Ability to [display the Model 204 Universal Buffer](#)^[123]

At least version 7.3 of the *Sirius Mods* is required.

- Ability to [display the value of a Model 204 parameter](#)^[122]

At least version 7.3 of the *Sirius Mods* is required.

Build: 34 (20 June 2008)

▣ These are the principal changes since the previous build:

- New **Execution Menu** option (**Get History**) and mappable command (getHistory) for [displaying a history of the statements executed](#)^[134] to the current point in a program being debugged

At least version 7.3 of the *Sirius Mods* is required.

Build: 33 (30 April 2008)

▣ These are the principal changes since the previous build:

- User items specified during Client session now are preserved in the next session:
 - Previous search strings are accessible in a **Search** box drop-down list.
 - Previous variables specified are accessible in a **Variable or Field** box drop-down list.
 - **Watch Window** contents are [redisplayed](#)^[91] by default.
 - Resizing of the Client window is preserved.
- **Watch Window** operations are no longer prevented when no program is being debugged
- New DebuggerTools class method, [Command](#)^[164], runs SIRIUS DEBUG subcommands within a request

At least version 7.3 of the *Sirius Mods* is required.

Sirius Mods Versions 7.0 - 7.2

Build: 32 (18 February 2008)

- ☐ These are the principal changes to the Debugger Client since the previous build:
 - A fix for a rare case where the **Watch Window** failed to display certain variables in a program involving a persistent session

Build: 31 (08 February 2008)

- ☐ These are the principal changes since the previous build:
 - Enhancements to [Source Preview](#)^[85] feature:
 - A Step or Search command is allowed and causes a download of the entire source program
 - A Run command runs the program without a code download
 - A fix for the case where the Client was failing to display a compilation error message (M204.1265) for which a *Model 204* MSGCTL was being issued

Build: 30 (28 January 2008)

- ☐ These are the principal changes since the previous build:
 - New option that allows [previewing of program code](#)^[85]
At least version 7.2 of the *Sirius Mods* is required.
 - Array item references that have spaces between the array name and the left parenthesis are now recognized properly by all the Client features

Build: 29 (09 December 2007)

- ☐ These are the principal changes since the previous build:
 - Displaying in the **Audit Trail** tab the names of procedures excluded from debugging by the [White List processing](#)^[79] feature
At least version 7.2 of the *Sirius Mods* is required.
 - A command for [interrupting](#)^[82] White List or Run Until processing
At least version 7.2 of the *Sirius Mods* is required.
 - Syntax added for watching mixed-case [global variables](#)^[96]

- Changes to the [Search feature](#)^[39] to enable keyboard-invoked consecutive searches for a given string:
 - The default response to pressing the Enter key while the **Search** text box has focus is changed from "search from the top" to "search from current line." This lets you press Enter repeatedly to find subsequent occurrences of a search string. Formerly, pressing Enter again found the same occurrence as the first time because it repeated the search from the top.

Note: This introduces a small upward incompatibility.
 - New **focusToSearchBox** command gives the input focus to the Search string text area. The Ctrl+F key combination is changed to perform the same function by default.

Build: 28 (09 November 2007)

☐ These are the principal changes since the previous build:

- A [console](#)^[316] for macro information
- **Macro Command Line** [menu option](#)^[36] to run a macro
- **Echo** command output goes to [console](#)^[316] if console is open
- A ... **macro running** indicator to be displayed on the Client's title bar
- New macro-related commands: [set](#)^[255], [assert](#)^[180], and [vardump](#)^[280]
- New mappable commands:
 - [addWatchOnCurrentLine](#)^[180] (with accompanying menu item **Add Watch on Current Line**)
 - [toggleBreakpointOnCurrentLine](#)^[272] (with accompanying menu item **Toggle Breakpoint on Current Line**)
 - [reloadWhiteList](#)^[238], [turnOnWhiteList](#)^[277], and [turnOffWhiteList](#)^[276] (with accompanying menu items **Reload White List**, **Turn On White List**, and **Turn Off White List**)
- New menu item (**Execution > Run Until Proc**) for Run Until processing, and the [runUntil command](#)^[245] is now mappable
- Lines located via a Search button are now highlighted and displayed in the center of the page

Build: 27 (27 September 2007)

☐ These are the principal changes since the previous build:

- [Jump feature](#)^[83] is added
At least version 7.2 of the *Sirius Mods* is required.
- New Client macro-related command ([echo](#)^[203])
- [&argstring macro variable](#)^[314] for passing arguments to macro commands
- [&&prompt function](#)^[315] for interactive prompting
- An exception to ordinary request-cancellation handling for the purpose of [ON UNIT code debugging](#)^[139]
- Search buttons now located below the text box for search strings

Sirius Mods Versions 7.0 - 7.1

Build: 26 (17 August 2007)

☐ These are the principal changes to the Debugger Client since the previous build:

- A new utility for [user-created macros](#)^[309]
- File types the *Janus Debugger* will *not* debug are [settable](#)^[376]

Build: 25 (13 July 2007)

☐ These are the principal changes since the previous build:

- New Client [Preferences option](#)^[18] (**Trim blanks from selection**) to remove blanks from text copied to the [Text Viewer](#)^[149].
- Support for additional level of Client menu and keyboard reconfiguration via the [uimore.xml](#)^[291] file
- [Expanded menu bar](#)^[17], which accommodates all Client commands

Build: 24 (29 June 2007)

☐ These are the principal changes since the previous build:

- [help](#)^[212] and [manual](#)^[224] programmable commands simplify access to product documentation

Build: 23 (27 June 2007)

☐ These are the principal changes since the previous build:

- [turnOffDebugging](#)^[276] programmable command turns off debugging from the Client
At least Client build 23 and version 7.1 of the *Sirius Mods* are required.
- Debugger now recognizes %variable names that include question marks (?) and single-quotation marks (')
At least version 7.1 of the *Sirius Mods* is required.
- Screen and ScreenField objects are added to the list of [viewable](#)^[111] classes
At least version 7.1 of the *Sirius Mods* is required.

Build: 22 (20 June 2007)

☐ These are the principal changes since the previous build:

- Two new [SIRIUS DEBUG commands](#)^[151] (SUSPEND and RESUME)
At least version 7.1 of the *Sirius Mods* is required.
- NamedArrayList objects are added to the list of [viewable](#)^[111] classes

Build: 21 (13 June 2007)

☐ These are the principal changes since the previous build:

- Client's **Web Buffer** tab displays output if you [debug a web thread program](#)^[157] that contains an embedded SIRIUS DEBUG ON command
- **File** menu option to [create or edit the ui.xml file](#)^[18], which contains button and hot key reconfigurations
- [Settable](#)^[124] image/screen items
- **Help** menu **Keyboard Shortcuts** display can be printed and saved

Build: 20 (08 June 2007)

☐ These are the principal changes since the previous build:

- [Viewing the Variables](#)^[111] in a Sirius or user-defined class
- Print options added to [Value window](#)^[101] **File** menu
- New DebuggerTools method ([DebugOff](#)^[165]) mimics **SIRUS DEBUG OFF** command

At least version 7.1 of the *Sirius Mods* is required.

Sirius Mods Version 7.0

Build: 19 (26 April 2007)

☐ These are the principal changes to the Debugger Client since the previous build:

- Fix provided for problem with requests longer than 64, 000 lines

Build: 18 (20 April 2007)

☐ These are the principal changes since the previous build:

- **Text Viewer** window for [copying, printing, or saving text](#)^[149] data from Client pages
- The button whose functionality [the Enter key repeats](#)^[291] is highlighted by a white background

Build: 17 (30 March 2007)

☐ These are the principal changes since the previous build:

- Settable buttons and hot keys
You may [reconfigure](#)^[283] any of the ten contiguous buttons below the main window to invoke a different feature than its default. You also may create your own keyboard shortcuts for commonly used features.
- From **Help** menu option, [keyboard shortcuts](#)^[286] can be displayed
- [Trace button](#)^[48] has a new default hot-key combination: Ctrl+T
- Enter key is alternative to the **Set** button in the [Set dialog box](#)^[124] for changing a variable's value
- Cosmetic changes to buttons

The button widths are now uniform, their labels have a smaller font, and disabled buttons are more obviously so.

Build: 16 (05 March 2007)

☐ These are the principal changes since the previous build:

- Name of current procedure [displays in GUI title bar](#)^[387]
- Workstation IP number displays on startup

The **Audit Trail** page [initially displays](#)^[37] a line like the following:

Local IP address 198.242.244.234

Build: 15 (22 February 2007)

▣ These are the principal changes since the previous build:

- Pressing Enter key repeats last GUI action

Pressing the Enter key has the same effect as clicking the [button that is currently highlighted](#)^[55]. For example, when a request is first presented in the **Source Code** tab, the **Step** button is highlighted. You can step through the program simply by pressing Enter repeatedly.

Build: 14 (19 February 2007)

▣ These are the principal changes since the previous build:

- Items in an array of \$lists can be [displayed in Value window](#)^[104]

In this context, subscripts are allowed for \$list identifiers. For example:

list %alpha(%i)

Build: 13 (09 February 2007)

▣ These are the principal changes since the previous build:

- Regular expressions supported in [the Search box](#)^[39]
- [SIRIUS DEBUG ON](#)^[151] command enhancement simplifies its syntax

Two command parameters are settable in the User 0 stream. At least version 7.0 of the *Sirius Mods* is required.

- Options to [set multiple breakpoints at once](#)^[60]— on all lines that match a search string or regex, or on executable statements that follow comment lines that begin with ***Break**

At least version 7.0 of the *Sirius Mods* is required.

- [Breakpoint](#)^[57] maximum increased from 40 to 1000 per User Language request

At least version 7.0 of the *Sirius Mods* is required.

- In FEO loops, [display current occurrence](#)^[116] value

At least version 7.0 of the *Sirius Mods* is required.

Build: 12 (22 January 2007)

☐ These are the principal changes since the previous build:

- Additional way to [display the items in a \\$list or Stringlist](#)^[104] variable
You can now right-click the variable's name in the **Watch Window** display. This is available for *Sirius Mods* versions 6.9 and 7.0 and higher.
- [Copy page contents](#)^[10] to clipboard
You can press the Ctrl+C keyboard key combination to copy the contents of the active (topmost) tab to the Windows clipboard.
- Additional "trace until" option
The **Run To Change** button continues request execution until the value of a selected variable changes. The new variation to this feature (press Alt key while clicking **Run To Change**) continues execution unless or until a selected variable's value becomes equal to a value you specify.
[Tracing until a specific value](#)^[133] is available only for *Sirius Mods* version 7.0 and higher.
- Length limit increased to 255 for %variables whose value you [set](#)^[124]
This is available for *Sirius Mods* versions 6.9 and 7.0 and higher.

Build: 11 (09 January 2007)

☐ These are the principal changes since the previous build:

- STATUS option for SIRIUS DEBUG command
Using [the STATUS option](#)^[151] produces a simple status report of the Debugger worker threads. At least version 7.0 of the *Sirius Mods* is required.
- White List processing enhancements
Processing is moved to the mainframe to eliminate code transfer and provide response time benefit. Also, [trailing wild cards](#)^[79] are now allowed in the White List.
- Step Out option
By pressing the Alt key and clicking the Client GUI **Step Over** button, you can immediately discontinue the Debugger processing of a called subroutine, method, or daemon and continue processing at the statement after the call to the subroutine, method, or daemon. This action is called a ["Step Out"](#)^[64].
- Skipping over Daemon code

By pressing the Alt key and clicking the Client GUI **Run** button, you can [discontinue the interactive debugging](#)^[142] of the Daemon object calls in your program. The Daemon code executes but is not displayed in the Debugger Client.

- Assembler language replaces User Language for Debugger internals
 - Downloading a User Language dump file (SIRDEBUG) to install the Debugger Server is no longer necessary.
 - The Windows Installer program (setup.exe) for the Debugger workstation client, which is stored in the SIRDEBUG procedure file for the *Sirius Mods* prior to version 7.0, is downloadable from the Sirius web site for Debugger customers running *Sirius Mods* 7.0.
- Removal of [requirements](#)^[365] to license *Janus SOAP* and *Janus Sockets* for the Debugger for users running version 7.0 of the *Sirius Mods*

A new Debugger Server port type (DEBUGGERSERVER) replaces the SRVSOCK type that is required for earlier versions of the *Sirius Mods*, and a new port type (DEBUGGERCLIENT) replaces the CLSOCK type that is required for the *Sirius Debugger* for earlier versions of the *Sirius Mods*.

- New **File > Preferences** [option](#)^[62] lets you break Debugger execution after READ SCREEN/MENU statements

New message (**Full Screen Read Pending**) prompts *Sirius Debugger* Client users when READ SCREEN is active. Formerly, the message was **Waiting for Online**.

- [Breakpoint](#)^[57] maximum increased from 20 to 40 per User Language request
- Searching from bottom to top of page is added (via Alt key) to [Search and Search Next](#)^[39] buttons
- After failed compilation, [F11 and F10 keys](#)^[138] allow navigation to next and previous statement, respectively, that did not compile
- [Watching variables within a class definition](#)^[100] no longer requires explicit **%this** specification

Sirius Mods Versions 6.8 and 6.9

Build: 10 (04 December 2006)

▣ These are the principal changes to the Debugger Client since the previous build:

- Option to restart the Debugger Client

From the file menu, the [Restart option](#)^[18] shuts down the Debugger Client. The Client then restarts, doing the same processing as if you started it by clicking its desktop icon.

- Main working area is resizeable

You can now drag the black bar just above the page-navigation buttons to resize the main working window in the Client.

- Improvements to \$list and Stringlist viewing

For a float variable \$list handle or for a Stringlist object variable, you can now [request a display](#)^[104] of its elements, which will be presented in a data viewer similar to the way XML documents are displayed.

Build: 9 (17 November 2006)

☐ These are the principal changes since the previous build:

- Run Until spans debug sessions

The **Run Until spans HTTP requests** option is renamed to **Run Until spans debug sessions**, and it is generalized to work with the *Sirius Debugger* as well as with the *Janus Debugger*. With this option turned on, Run Until continues searching through the source program until it finds the specified procedure, even if the debugging session is interrupted by a loss of the connection to the Online or the *Sirius Debugger* is toggled off and on again. When the session resumes, the Client keeps searching for the target procedure.

- [Performance statistics](#)^[359] kept in log file are enhanced to report the number of bytes received per message incoming to the Debugger Client.

Build: 8 (13 November 2006)

☐ These are the principal changes since the previous build:

- Right-clicking **Source Code** lines lets you display variable values

As described in [Displaying the value of a program data item](#)^[101], this approach lets you display a variable's value just as if you entered the variable name in the **Variable or Field** input area, then clicked the **Value** button.

Build: 7 (5 November 2006)

☐ These are the principal changes since the previous build:

- The installation kit now installs a required MicroSoft file (gdiplus.dll) to the installation target folder instead of to the system root folder, avoiding any security issues concerning folder write permissions.
- Minor memory-use and performance improvements for both source-code loading and variable watching.

Build: 6 (25 October 2006)

☐ These are the principal changes since the previous build:

- Save and restore proxy settings for Microsoft Internet Explorer

The [automatic maintenance of proxy settings](#)^[382] for *Janus Debugger* IE users that was added in Build 5 did not save and restore *all* proxy-related settings. It now does.

- Performance improvements for watching variables

The display of the values of watched variables is slightly accelerated.

Build: 5 (18 October 2006)

☐ These are the principal changes since the previous build:

- Better proxy control for Internet Explorer

For those *Janus Debugger* users whose browser is Microsoft Internet Explorer, the task of defining the Debugger Client as the proxy server can be [handled automatically](#)^[382] by the Client when it starts up (and undone when the Client shuts down).

- Improved source line scrolling

To avoid having the current execution position in a Client tabbed-page window you are stepping through be the last line on the page (obscuring the next line of code), the display of the highlighted current position now includes the next two lines.

Build: 4 (2 October 2006)

☐ These are the principal changes since the previous build:

- Performance improvements for very long programs

Elapsed time for transferring and displaying programs with thousands of lines of code has been reduced by 90%. In addition, a new configuration file setting lets you trace the time the Client spends handling program code (see [Tracking Client performance](#)^[359]).

Build: 3 (27 September 2006)

☐ These are the principal changes since the previous build:

- **Run Until Procedure** processing can ignore the end of an HTTP request

This feature is designed to make it easier to debug HTML frame-based web applications, since the Debugger Client by default terminates the search for a designated Run Until procedure at the end of an HTTP request. Thus, a procedure search stops by default after each User-Language produced frame in a single HTML frameset statement.

The **File > Preferences** menu option displays a checkbox that lets you direct the Client to continue across multiple HTTP requests until the specified Run Until procedure name or pattern is satisfied. See [Precedence and scope for Run Until](#)^[77].

- Selecting which of multiple web servers are eligible for debugging

After selecting the **File > Preferences** menu option, you can choose which of the Onlines specified in the Client configuration file (`debuggerconfig.xml`) are to have their web requests debugged. For more information, see [Debugging multiple Web Servers](#)^[147].

- New hot-key combination for **Preferences** option

The Ctrl+P key combination now invokes the **Preferences** option of the **File** menu.

Build: 2 (4 September 2006)

▣ These are the principal changes since the previous build:

- Simplified editing and reloading of a White List

You can now edit or create a white list file (`whitelist.txt`) with the Windows *Notepad* editor by selecting **Edit White List** from the Debugger Client **File** menu. In addition, instead of restarting the Client to reload the list when done editing, you simply click the **Reload White List** button on the **Proc Selection** tab.

White list processing is described in [Running only to listed procedures](#)^[79].

- Enhanced status reporting
 - An hourglass cursor displays whenever the Client is waiting for processing to complete on the Online.
 - Additional [status area](#)^[50] message values indicate that the GUI is waiting for the Online to respond:
 - **Waiting for Online** is displayed when source code is being sent from the Online to the Client workstation, or when the Online is processing the Client's **Step**, **Run**, **Run Until**, or **Trace** command. The **Waiting for Online** message is also displayed in *Sirius Debugger* sessions whenever the Client awaits a new request from the Online.
 - **Receiving/Forwarding Web Page** is displayed by the *Janus Debugger* Client when a Web Server's HTTP response to a web request is being read by the Client and forwarded to the web browser.

- **Persistent Session Suspended** (see [Debugging Web Server persistent sessions](#)^[144]) is changed to **Session awaits browser**.
- When White List processing is toggled on or off from the **Proc Selection** tab, **White list is active** or **White list turned off** is displayed as appropriate.
- New hot-key combination for **Search** command
The Ctrl+F key combination now performs the same function as the **Search** button (see [The navigation and execution buttons](#)^[39]).
- A summary of the changes since the last build
As described in [Versions and builds](#)^[6], you can view a list of the changes in the current build of the Client by selecting **About** from the Debugger Client **File** menu.

Index

!

!debugger exclude off statement 67
 !debugger exclude on statement 67
 !debugger include off statement 67
 !debugger include on statement 67
 !debugger statements 67, 260

\$

\$COMMBG requests 2
 asynchronous 6
 \$CURREC function 87, 97
 \$FIELDGROUPLD function 87, 121
 \$FIELDGROUPOCCURRENCE function 87, 121
 \$GETG function 96
 \$list 205
 array 87, 98, 104
 displaying 104
 watching 87, 98
 \$listcnt function 98
 \$listinf function 98
 \$STATUS function 87, 97
 \$STATUSD function 87, 97
 \$WEB_FORM_DONE requests 2, 144

%

%this keyword 100

&

&&amDaemon function, Debugger Client 322
 &&arg function, Debugger Client 314, 315, 323
 &&assertFailureCount function, Debugger Client 180, 241, 324
 &&assertStatus function, Debugger Client 180, 324
 &&assertSuccessCount function, Debugger Client 180, 241, 325
 &&blackOrWhiteList function, Debugger Client 326

&&concatenate function, Debugger Client 327
 &¤tRunningMacro function, Debugger Client 328
 &¤tTitle function, Debugger Client 330
 &&exists function, Debugger Client 331
 &&functions 321
 &&getMainSearchInputArea function, Debugger Client 332
 &&getVariableOrFieldInputArea function, Debugger Client 332
 &&globalAssertFailureCount function, Debugger Client 180, 241, 332
 &&globalAssertStatus function, Debugger Client 180, 241, 333
 &&globalAssertSuccessCount function, Debugger Client 180, 241, 333
 &&iemode function, Debugger Client 333
 &&index function, Debugger Client 334
 &&isWatched function, Debugger Client 334
 &&length function, Debugger Client 335
 &&numberOfBreakpoints function, Debugger Client 335
 &&numberOfLevels function, Debugger Client 336
 &&numberWatched function, Debugger Client 336
 &&originalTitle function, Debugger Client 337
 &&preference function, Debugger Client 337
 &&procName function, Debugger Client 338
 &&prompt function, Debugger Client 315, 338
 &&searchResult function, Debugger Client 339
 &&searchSuccess function, Debugger Client 339
 &&selectedTab function, Debugger Client 340
 &&statusMessage function, Debugger Client 340
 &&substring function, Debugger Client 341
 &&sum function, Debugger Client 341
 &&verifyMatch function, Debugger Client 342
 &&verifyNoMatch function, Debugger Client 343
 &&windowStatus function, Debugger Client 343
 &argstring variable, Debugger macro 314

*

*Break comment lines 60

.macro file extension 297, 309
.NET Framework 369
.watch file extension 91, 220, 246, 297

?

?& dummy strings 127

3

3270 Emulator text box 63
3270 Emulator, Preferences option 18

A

About the Debugger Client box 6
About window 266
About, menu option 6, 39, 266, 395
about.xml file 297
Account, UltraEdit FTP 173, 175
Add Watch on Current Line, menu option 33, 88
Add Watch, menu option 15, 33, 88
addWatch command 180
addWatchOnCurrentLine command 180
Alt + Run combination 142
Alt + Run To Change combination 134
Alt + Search combination 45, 128
Alt + Search Next combination 46
Alt + Step Over combination 64
Alt key 191, 224, 226, 286
Alt+B key combo 60, 291
Alt+F10 key combo 64, 291
Alt+F5 key combo 142, 291
Alt+F9 key combo 46, 291
AmDebugging method, DebuggerTools class 162
APSY subsystem 126
architecture, Debugger 4
Arraylist object variables 98, 104, 113, 205
arrays
 \$list 87
 %variable 87
assert command 180, 316
audit trail 14, 128
Audit Trail tab 10, 40
AutoRun, Macro 260, 318

B

background color 18, 295
Background color panel 295
backslash, escape character 45, 247, 248, 250, 251
Binary button 93, 102, 123
binary value, display of 93, 102
Black list is active, message 81
Black List processing 13, 79
Black list reloaded message 82
blacklist.txt file 79, 238
blanks, removal of 18, 149
Boolean enumeration variables 124
Bottom button 40, 301
bottom command 183, 291
Bottom option, Preferences window 41
Bottom, menu option 24
Break after READ SCREEN, Preferences option 18, 62, 260
Break Background color panel 295
Break method, DebuggerTools class 162
Break on next proc set message 78, 82
Break Text color panel 295
breakAfterReadScreen option, setPreference command 260
breakOnNextProc command 78, 82, 184
Breakpoint cleared 60
Breakpoint set 58
breakpoints 27, 57, 58, 60, 61, 162, 188, 257, 272, 335
Breakpoints cleared 40
Breakpoints menu 27
Breaks At, menu option 27, 60
breaks command 60, 184, 291
Breaks, menu option 27, 60
breaksAt command 60, 185, 291
browser, web
 communication error 356
 configuration 380
 in Debugger architecture 4
 surfing while debugging 381
buffer, web output 129
builds, Debugger Client 6, 266, 395
button attribute, ui.xml file 287
button bar 26, 39, 224, 300
 Dock menu 42, 186, 207
 external window 42, 186, 207, 307
 extra 21, 43, 224, 285, 287
 File menu 42, 186, 207

button bar 26, 39, 224, 300
 position 41, 223
 button modifier 224, 286
 button, named 285
 button, separator 224, 285, 288
 buttonBar command 26, 42, 186
 buttonModifier attribute, ui.xml file 288
 buttons 224
 common 39, 44
 default settings 291
 highlighted 55
 program execution 40
 reconfiguring settings of 283

C

Cancel button 40, 65
 cancel command 187, 291
 Cancel Errors color panel 295
 Cancel, menu option 29
 Cannot find proxy server 357
 caseSensitiveAssert option, setPreference command 260
 Center option, Preferences window 41
 Change Value option 124
 CharacterMap object, Sirius 113
 characters, printable 129
 CharacterTranslationException object, Sirius 113
 Chrome browser 381
 class member Variables 100, 111, 211
 Clear All Breakpoints, menu option 27, 61
 Clear Audit button 40, 128
 Clear Audit Trail, menu option 24
 Clear Breaks button 40, 61
 Clear Execution Trace, menu option 24
 Clear IE proxy override 361, 385
 Clear IE proxy override, Preferences option 19
 Clear Watch button 40, 90
 Clear Watch, menu option 33
 Clear Web Buffer, menu option 12, 24
 clearAudit command 187, 291
 clearBreakpointOnCurrentLine command 60, 188
 clearBreaks command 61, 189, 291
 clearButton command 189, 284
 clearExecutionTrace command 190
 clearHistory command 134, 190
 clearKey command 191, 284
 clearMacroConsole command 191, 316
 clearStatus command 192

clearWatch command 90, 192, 291
 clearWebBuffer command 193
 clearWebBuffer_cmd 12
 Client, Debugger
 See Debugger Client 2
 ClientCommand method, DebuggerTools class 164, 179
 CLIENTCOMMAND option, SIRIUS DEBUG command 151, 155, 164
 Close External Windows, menu option 24
 closeCommandLine command 193
 closeExternalAuditTrailWindow command 193
 closeExternalButtonWindow command 194
 closeExternalButtonWindows command 42
 closeExternalExecutionTraceWindow command 194
 closeExternalWatchWindow command 194, 300
 closeExternalWebBufferWindow command 195
 closeExternalWindows command 24, 195
 closeHistory command 134, 195
 closeMacroConsole command 196, 316
 closeValueDisplay command 196
 code preview 18, 85
 collectTuningData element, configuration file 359
 Color Preferences 18
 Color Preferences window 293
 Color Preferences, menu option 293
 color, text 18, 293
 command attribute, ui.xml file 288
 Command keyword 155, 162, 164
 command line 317
 command line, Client 179, 317
 Command Line, dialog box 193, 231, 317
 Command Line, menu option 36, 179, 193, 231, 317
 Command method, DebuggerTools class 164
 Commands window 38
 commands, Debugger Client 179, 191, 224, 226, 285, 291
 macro-only 179, 197, 198, 201, 205, 214, 215, 222, 223, 229, 260, 269
 within a macro 309, 314, 315
 Commands, menu option 38, 266
 Comments color panel 295
 Communication Error message 354, 357, 387
 Compile Errors color panel 295
 Compile errors! message 138
 configuration

- configuration
 - browser 380
 - Debugger Client 374
 - proxy server 380
- configuration file, Debugger Client
 - See debuggerConfig.xml file 374
- Connection from Online message 152
- connection, SSL 159, 374, 387
- Console window 234, 316
- console, macro 316
- Console, menu option 36, 196, 221, 234, 316
- console.xml file 297
- Continue method, Daemon 143
- ContinueAsync method, Daemon 143
- continuel command 197, 198
- ContinueIndependently method, Daemon 143
- copy active display to clipboard 10, 24, 200
- copy command 200, 291
- copy cursor-selected content 149, 200
- copy entire tab to Text Viewer 10, 149
- Copy, menu option 10, 24
- Count method, Stringlist 98
- createMacro command 200
- CRLF parameter, Serial method 110
- Ctrl key 191, 224, 226, 286
- Ctrl+B key combo 60, 291
- Ctrl+C key combo 10, 291
- Ctrl+F key combo 39, 44, 128, 291
- Ctrl+P key combo 18, 291
- Ctrl+T key combo 48, 130, 291
- Ctrl+U key combo 39, 45, 128, 291
- Ctrl+X key combo 65, 291
- current keyword, JumpToLine command 216
- cursor, hourglass 388

D

- Daemon object, Sirius 113
- Daemon tab 141
- daemons 64, 141
- Data Display menu 33
- Dataset object, Sirius 113
- Debug Previewed Source, menu option 29, 86
- Debugger
 - authorization 365
 - debugging of 345, 348
 - directives 67, 260
 - documentation 7, 38, 379
 - overview of 2
 - prerequisites 365
 - starting 7

- Debugger Client 2
 - builds 6, 266, 395
 - configuration file 374
 - customization 283
 - file updates 38, 390
 - hiding lower windows 213, 243, 273, 305
 - installation 369
 - listening port 369
 - main window 14, 243, 264, 330, 337
 - release notes 395
 - restarting 18
 - user interface commands 179, 191, 224, 226, 285, 291
 - window transparency 307
 - work files 296, 378
 - workstation IP address 370, 388
- Debugger Client Update dialog box 390
- Debugger Server 345, 348
 - client socket port, Sirius Debugger 152, 368, 388
 - server socket port 152, 367
- DEBUGGERCLIENT port, Sirius Debugger 368
- debuggerConfig.xml file 18, 147, 170, 175, 371, 374, 390
- debuggerDirectives option, setPreference command 260
- debuggerPac.js 385
- debuggerPac.js file 209
- DEBUGGERSERVER port 367
- DebuggerTools class 161
- DEBUGMAX system parameter, Model 204 156, 366
- DebugOff method, DebuggerTools class 165
- DEBUGPAG system parameter, Model 204 366
- debugPreview command 86, 201
- DEBUGSERVER.UL procedure 345, 348
- decrement command 201
- details, procedure 126
- dir.txt file 371
- directives, Debugger 67, 260
- disableButton command 202, 284
- Display Options 18, 104
- Display Options, Preferences options 18
- Doc folder 371
- documentation, Debugger 7, 38, 371, 379
- documentationURL element, configuration file 379
- dummy strings 126, 127

E

- EBCDIC binary values 102
- echo command 203
- echo command 316
- Edit Black List button 79, 82
- Edit Black List, menu option 18, 79, 82
- Edit button 126
- Edit debuggerConfig.xml, menu option 18
- Edit Exclude Proc List button 70
- Edit Exclude Routine List button 70
- Edit Include Proc List button 70
- Edit Include Routine List button 70
- Edit Macro, menu option 36, 204, 309
- Edit ui.xml, menu option 18, 287
- Edit uimore.xml, menu option 22, 291
- Edit White List button 79, 82
- Edit White List, menu option 18, 79, 82
- Edit, menu option 36
- editing procedures 126
- editMacroFromUISelection command 204
- editor element, configuration file 175
- editor, Client text 166
- editor, local 166
- editor, source code 166
- editor, text
 - element, configuration file 379
- Email object, Sirius 113
- enableButton command 204, 284
- encrypted connection 159, 374, 387
- Enter key 39, 44, 46, 55, 56, 64, 128, 133, 291, 301
- Entity-name input box 15, 51, 95
- Error menu 35
- errors 35
 - communication 354, 357, 387
 - compilation 35, 138, 290
 - program 35, 138
 - request cancellation 139
- escape character 45, 247, 248, 250, 251
- evaluate command 205
- Evaluation successfully completed 55, 56
- exception classes, system 111
- Exclude directives 66, 67
- Exclude Parts of Program from Debugging 13, 67
- excludeProc.txt file 70, 239, 297
- excludeRoutine.txt file 70, 239, 297
- executed code
 - procedure details for 129
- Executed one statement 55
- Execution History window 18, 134
- Execution menu 29
- Execution Options 18
- Execution Options, Preferences options 18
- Execution Position color panel 295
- Execution Trace tab 13, 129
 - capacity of 6
 - for execution history 18, 134
- execution, program code 54
- Expand Object option 111, 206
- expandList command 104, 205
- expandObject command 111, 206
- expression, HTML or TEXT statement 96
- external window 24, 193, 194, 195, 231, 232, 233, 236, 278, 281, 300, 303, 343
- Extra Buttons window 43, 207
- Extra Buttons, Preferences option 21, 44
- extraButtonBar command 207
 - main parameter 44

F

- F1 key 7, 38
- F10 key 40, 56, 138, 291
- F11 key 40, 55, 138, 291
- F4 key 39, 40, 55, 291
- F5 key 39, 40, 56, 291
- F9 key 46, 128, 291
- FastUnloadTask object, Sirius 113
- FEO (FOR EACH OCCURRENCE OF) statement 116, 208
- FEO OCC IN value 116, 208
- feoDisplay command 116, 208
- field groups, Model 204 33, 95, 120, 234
- fields, Model 204 87, 95
- File menu 18
- file types, debugger-ignored 260, 376
- file, PAC 209
- file, procedure 126
- filter element, configuration file 376
- find.txt file 297
- Firefox browser 382
- firstHistory command 30, 136, 208
- FloatNamedArraylist object, Sirius 113
- focusToSearchBox command 209, 291
- font size 299, 378
- fontScale element, configuration file 299, 378
- frames, HTML 77
- FTP Account, UltraEdit 173
- FTP Server, Janus 172

Full Screen Read Pending message 62
function keys 191, 226
Functions window 38
functions, Debugger Client 321
 string rules 322
Functions, menu option 38, 267

G

generatePac command 209
Get It button 390
Get/Display History, menu option 29, 134
getHistory command 134, 211
getVariableList.xml file 297
getVariablesForClass command 111, 211
global variables 87, 96, 127

H

help command 212
Help menu 38
Help Topics, menu option 38
Help, online 38
 printing 7
Hide Lower Section, Preferences option 21,
213, 243, 273
hideLower command 21, 213, 305
highlight color 18, 293
History to Execution Trace, Preferences option
18, 134, 260
history, statement 18, 29, 134, 136, 208, 211,
220, 228, 237
history.xml file 297
historyToTrace option, setPreference command
260
Honor Line Ends button 101, 123
 value of watched item 93
host, Model 204 Online 173, 374
hot keys 291
 Alt+B key combo 60, 291
 Alt+F10 key combo 64, 291
 Alt+F5 key combo 142, 291
 Alt+F9 key combo 46
 Alt-F9 key combo 291
 Ctrl+B key combo 60, 291
 Ctrl+C key combo 291
 Ctrl+F key combo 44, 128, 291
 Ctrl+P key combo 18, 291
 Ctrl+T key combo 48, 130, 291
 Ctrl+U key combo 45, 128, 291
 Ctrl+X key combo 65, 291

 default settings 291
 Enter key 44, 46, 55, 56, 128, 133, 291
 F1 key 7, 18, 38
 F10 key 39, 40, 56, 138, 291
 F11 key 39, 40, 55, 138, 291
 F2-F12 191, 226
 F4 key 39, 40, 55, 291
 F5 key 39, 40, 56, 291
 F9 key 46, 128, 291
 reconfiguring settings of 283
hourglass cursor 388
HTML output 12
HTML statement, User Language 96, 295
HTTP Helper 160
HttpRequest object, Sirius 113
HttpResponse object, Sirius 113
HTTPS protocol 159, 387

I

IE Mode, Preferences option 19, 258, 383, 385
IE Options, Preferences options 19, 383
ieAuto option, setPreference command 260
ignoredFileTypeList option, setPreference
command 260, 376
ignoreMacroErrors option, setPreference
command 260
image items 87
In window, command prefix 44, 183, 247, 248,
250, 251, 274, 288
Inactive color panel 295
include command 66, 214
include command, Debugger macro 309
Include directives 66, 67
INCLUDE statement 66
includelf command 214
includeProc.txt file 70, 239, 297
includeRoutine.txt file 70, 239, 297
increment command 215
Indent parameter, Serial method 110
Init Exclude 74
initExclude option, setPreference command
260
inner procedure 66
installation folder 296, 371, 378
installation, product 363
Internet Explorer browser 260, 382
Invalid Jump message 83
Invalid line for jump 84, 217
InvalidBase64Data object, Sirius 113
InvalidHexData object, Sirius 113

InvalidRegex object, Sirius 113
 IP address
 Debugger Client workstation 370, 388
 Model 204 host 172, 173
 IPCONFIG command, DOS 160, 370
 Item method, Stringlist 98

J

JANUS CLSOCK ALLOW command 368
 JANUS commands, for FTP Server 172
 Janus Debugger
 overview of 2
 starting 387
 Janus FTP Server 172
 Janus Network Security 159, 374
 Janus Web Legacy Support sessions 63, 146
 Janus Web Server
 debugging multiple instances of 147, 374
 port number 374
 SSL port 159, 374, 387
 Janus/Sirius Debugger User's Guide 7, 38
 JanusDebugger.chm file 371
 JanusDebugger.exe file 371, 390
 jdebugr.pdf file 371
 Jump Here option 83
 jump operation 83, 216, 217
 jumpToLine command 84, 216
 jumpToMatch command 84, 217

K

KEEPALIVE parameter, JANUS DEFINE 368
 key attribute, ui.xml file 288
 key modifier 191, 226, 286
 keyboard shortcut 38, 179, 191, 226, 286
 Keyboard Shortcuts window 38
 Keyboard Shortcuts, menu option 38
 keyModifier attribute, ui.xml file 288
 keys, shortcut 38, 291
 Alt+B key combo 60, 291
 Alt+F10 key combo 64, 291
 Alt+F5 key combo 142, 291
 Alt+F9 key combo 46
 Alt-F9 key combo 291
 Ctrl+B key combo 60, 291
 Ctrl+C key combo 291
 Ctrl+F key combo 44, 128, 291
 Ctrl+P key combo 18, 291
 Ctrl+T key combo 48, 130, 291
 Ctrl+U key combo 45, 128, 291

Ctrl+X key combo 65, 291
 default settings 291
 Enter key 44, 46, 55, 56, 128, 133, 291
 F1 38
 F1 key 7
 F10 key 39, 40, 56, 138, 291
 F11 key 39, 40, 55, 138, 291
 F2-F12 191, 226
 F4 key 39, 40, 55, 291
 F5 key 39, 40, 56, 291
 F9 key 46, 128, 291
 reconfiguring settings of 283
 kill command 218
 Kill Running Macro, menu option 36, 309

L

labelButton command 219, 284
 lastHistory command 31, 136, 220
 LAUDPROC parameter 76
 Ldap object, Sirius 113
 Legacy Support sessions 63, 146
 limitations, product 6
 line number, procedure 126
 line number, source code 11
 line type, source code 11
 line-end characters 101
 in serialized output 106
 printable 129
 watched item 93
 List Display option 104, 205
 Load Watch, menu option 33, 91
 loadWatch command 220
 LOB data 6
 log, client 348, 353
 log.txt file 297, 348, 353
 LPDLST parameter 366, 367
 LSTBL parameter 366
 LVTBL parameter 366
 Lynx browser 386

M

Macro Autorun, Preferences option 18, 260, 318
 macro command 221, 309, 314
 Macro Command Line, dialog box 193, 231, 317
 Macro Command Line, menu option 36, 193, 231, 309, 317
 Macro completed 309

Index

Macro Console window 196, 221, 234, 316
Macro Console, menu option 36, 234, 309, 316
Macro keyword 155, 162, 164
 command line 317
Macro prompt 315
macro, Debugger 36, 309, 314
 argument variable 314
 autorun 260, 318
 command line 317
 commands 179, 197, 198, 201, 205, 214, 215, 222, 223, 229, 260, 269
 console 316
 functions 321
 variables 271, 280, 319
macroAutorun option, setPreference command 260
MacroConsole command 221
macroLibraryFolder element, configuration file 296, 378
macro-only commands 179, 197, 198, 201, 205, 214, 215, 222, 223, 229, 260, 269
Macros menu 36
MacroTrace command 222
macroWait command 223
Main Button Bar area, Preferences window 41
Main Button Bar, Preferences options 18, 21
main parameter, extraButtonBar command 44
Main Window Options, Preferences option 18
Main Window Options, Preferences options 21, 213, 243, 273
mainButtonBar command 223
mainline code 64
manual command 224
mapButton command 224, 284
mapKey command 284
mapping element, ui.xml file 288
mappings element, ui.xml file 287
MaxDaemExceeded object, Sirius 113
menu bar, Client 17
menus, Client 17, 179
mergedPac option, IE Mode 385
methods, DebuggerTools class 161
Minimum program length, for preview 85
Model 204
 fields, watching 95
 parameter values 122
 Universal Buffer 123
modifier, button 224, 286
modifier, key 191, 226, 286
Most Recent Audit Trail window 14, 128

moveBrowserToTop command 227
moveTn3270ToTop command 227
msvcr71.dll file 371
multiply occurring fields 95

N

named button 285
NamedArraylist object, Sirius 113
NETSTAT command, DOS 370
New Blank Macro, menu option 36, 309
New Macro, menu option 36, 200
newPac option, IE Mode 385
Next Compile Error, menu option 35, 138
nextCompileError command 138, 228, 290, 291, 313
nextHistory command 30, 136, 228
NoFreeDaemons object, Sirius 113
none option, IE Mode 260, 383
noSpan command 229
NotePad++ editor 166, 379
notepadReplacement element, configuration file 166, 379
nsLookup command 230

O

object variables, Janus SOAP 97, 106, 111
OCC IN phrase 116, 208
OFF option, SIRIUS DEBUG command 151, 154, 155, 156
ON option, SIRIUS DEBUG command 151, 152, 156, 157
ON UNIT debugging 139
Online has disconnected message 387
opacity element, configuration file 307, 379
Open at Startup, Preferences options 18, 21
Open External Audit Trail Window, menu option 24
Open External Button Window, menu option 26, 42
Open External Execution Trace Window, menu option 24
Open External Watch Window, menu option 15, 24, 33, 300
Open External Web Buffer Window, menu option 24
openCommandLine command 231
openExecutionTraceWindow command 232
openExternalAuditTrailWindow command 231
openExternalAuditTrail command 24

openExternalButtonWindow 42
 openExternalButtonWindow command 232
 openExternalExecutionTrace command 24
 openExternalWatchWindow command 24,
 233, 300, 305
 openExternalWebBuffer command 24
 openExternalWebBufferWindow command
 233
 openMacroConsole command 234
 Opera browser 387
 Org.Mentalis.Security.dll file 371
 Out of Scope color panel 295
 outer procedure 66
 output, HTML 12
 output, tracing 13

P

PAC (Proxy Auto Config) file 209, 258, 267
 PAFGI (Print All Fieldgroup Information)
 statement 120
 pafgi command 120, 234
 PAFGI, menu option 33
 page copying 10, 149
 PAI (Print All Information) statement 117
 pai command 117, 235
 PAI, menu option 33
 parameters, Model 204 87, 122
 Paste, menu option 24
 Path system variable, Windows 170, 175
 pattern, procedure name 75
 Pause at end of evaluation, Preferences option
 18, 61, 260
 pauseAtEndEval option, setPreference
 command 260
 performance, Debugger Client 359
 persistent sessions 144
 skipped over by Run Until 77
 pin command 236, 303
 Pin option, context menu 236, 303
 port number, TCP
 Debugger Client proxy 376, 380
 Debugger Server 366, 367, 374
 Debugger Server client socket 368
 Janus FTP Server 172, 173
 Janus Web Server 374
 port, client socket (Sirius Debugger) 368
 port, SSL 159, 374, 387
 preference option, help command 212
 preferences command 236, 291
 Preferences window, opacity of 307, 379
 Preferences, menu option 18, 41, 61, 62, 77,
 85, 90, 91, 104, 134, 147, 260, 305, 383
 preferences.xml file 297
 prefix
 for class member Variables 100
 for field names 95
 for global variables 96, 127
 for shared object variables 97
 preview, source code 18, 85
 Preview: bad compile, message 86
 Preview: good compile, message 86
 Previous Compile Error, menu option 35, 138
 previousCompileError command 138, 237,
 290, 291, 313
 previousHistory command 30, 136, 237
 Print button 149
 Print option, online Help toolbar 7
 Print option, text viewer 149
 Print options, Value window 93, 101, 104
 PRINT statement output 12
 printable characters 129
 Proc Selection tab 13, 66, 75
 procedure 79, 126
 Black List 79
 details 126
 editing 126
 inner/outer 66
 name, wilcards for 70
 name, wildcards for 76
 running execution until 75
 White List 79
 Procedure Information option 126
 for dummy strings 127
 for trace data 129
 Procedure Line Details dialog box 126, 166
 program execution buttons 40
 program execution, controlling 54
 Program Titles 18, 63, 144, 146
 Program Titles, Preferences options 18
 Proxy Auto Config (PAC) file 385
 proxy element, configuration file 376
 proxy option, IE Mode 260, 361, 385
 proxy server 260
 automatic maintenance 209, 258, 260,
 361, 383, 385
 bypassing of 260, 361, 383
 Client as 4, 258, 267, 345, 369
 for HTTP Helper 160
 setting up 374, 380
 troubleshooting 361, 385

pushdown list (PDL) 366, 367

Q

Quit, menu option 35

R

RandomNumberGenerator object, Sirius 113

READ MENU statement 18, 62

READ SCREEN statement 18, 62, 146

Receiving/Forwarding Web Page 425

Record object, Sirius 113

Recordset object, Sirius 113

RecordsetCursor object, Sirius 113

regex

See regular expressions 45

regular expressions 45

Release Notes 395

Reload Black List button 81, 82, 238

Reload Black List, menu option 29, 238

Reload Proc/Routine/Method Lists button 70, 239

Reload White List button 81, 82, 238

Reload White List, menu option 29, 238

reloadBlackList command 29, 82, 238

reloadLists command 70, 239

reloadWhiteList command 29, 82, 238

Remove option, watched item 90

removeCurrentWatch command 90, 240

request cancellation 65, 139

resetAssertCounts command 241, 324, 325

resetGlobalAssertCounts command 241, 332, 333

resizing, display window 10

restart command 23, 242

Restart with Default Window Size, menu option 18, 23

Restart, menu option 23

restartDefault command 23, 242

Restore watches on startup 91

Restore watches on startup, Preferences option 18

restoreLower command 243, 305

restoreTitle command 243

RESUME option, SIRIUS DEBUG command 151, 155, 156

ReturnToMaster method 143

Run button 40, 56

run command 56, 244, 291

Run Macro, menu option 36, 244, 309

Run method, Daemon 143

Run To Change button 48, 133

Run Until Proc, menu option 29, 75

Run Until Procedure button 75, 162

Run Until processing 66, 75

Run Until spans debug sessions 77

Run Until spans debug sessions, Preferences option 18

Run Until Variable Changes, menu option 29, 133

Run Without Daemons, menu option 29, 142

Run, menu option 29, 36, 56

RunAsync method, Daemon 143

RunIndependently method, Daemon 143

runMacroFromUISelection command 244

runUntil command 245

runUntilVariableChanges command 133, 245

runWithoutDaemons command 142, 246, 291

S

Save button 149

Save Watch, menu option 33, 91

saveWatch command 246

Screen object, Sirius 113

ScreenField object, Sirius 113

script, automatic configuration 383

SDAEMDEV parameter 366, 367

sdaemons 64, 141, 366, 367

SDEBGUIP parameter 152

SDEBWRKP parameter 152

Search button 44, 45, 128

Search Down button 301

Search Down, menu option 26, 46

Search From Bottom, menu option 26, 45

Search From Top, menu option 26, 44, 45

Search menu 26

Search Next button 44, 46, 128

Search Prev button 44, 46, 128

search string, for breakpoints 27, 60

Search Up button 46, 301

Search Up, menu option 26, 46

searchDown command 46, 247, 291

searchFromBottom command 45, 248, 291

searchFromTop command 44, 45, 250, 291

searchUp command 46, 251, 291

Select First History Line, menu option 30, 136

Select Last History Line, menu option 31, 136

Select Next History Line, menu option 30, 136

Select Previous History Line, menu option 30, 136

- selectAuditTab command 252
- selectExecutionTraceTab command 253
- Selection color panel 295
- selectNextTab command 253
- selectProcSelectionTab command 253
- selectSourceTab command 254
- selectWatchWindow command 254
- selectWebBufferTab command 254
- separator button 224, 285, 288
- separator keyword 224, 285, 288
- Serial method 106
- Serial method formatting 110
- Serialize option 106
- server element, configuration file 374
- server list 374
- serverList element, configuration file 374
- Session awaits browser 144
- session ID 345, 348
- Set %var dialog box 124
- set command 255, 320
- setBlackList command 256
- setBreakpointOnCurrentLine command 58, 257
- setIEmode command 258, 267
- setM204Data command 124, 260
- setPreference command 61, 62, 68, 134, 258, 260
- setStatusMessage command 262
- setTitle command 264, 330
- setWhiteList command 265
- shared object variables 97
- shortcut keys 38
 - Alt+B key combo 60
 - Alt+F10 key combo 64
 - Alt+F5 key combo 142
 - Alt+F9 key combo 46
 - Ctrl+B key combo 60
 - Ctrl+F key combo 44, 128
 - Ctrl+P key combo 18
 - Ctrl+U key combo 45, 128
 - Ctrl+X key combo 65
- default settings 291
- Enter key 44, 46, 55, 56, 128, 133
- F1 38
- F1 key 7
- F10 key 39, 40, 56, 138
- F11 key 39, 40, 55, 138
- F4 key 39, 40, 55
- F5 key 39, 40, 56
- F9 key 46, 128
- reconfiguring settings of 283
- shortcut, keyboard 286
- shortcuts.xml file 297
- Show at most n list items 104
- Show at most n list items, Preferences option 18
- Show Extra Button Bar Window, menu option 43, 44
- Show long watch values in a Tooltip 90
- Show long watch values in a Tooltip, Preferences option 18
- Show Main Button Bar in External Window, menu option 26, 42
- showAbout command 39, 266
- showCommands command 38, 266
- showFunctions command 38, 267
- showIE command 267
- showShortcuts command 268
- Since Last Resume color panel 295
- SIRIUS DEBUG command 7, 151, 157, 164, 165, 388
- Sirius Debugger
 - connection information 388
 - for Janus Web Server threads 157
 - overview of 2
 - resuming 151
 - starting 151, 388
 - stopping 151, 388
 - suspending 151
- Sirius Documentation, menu option 38, 379
- Sirius Mods versions 6
- Sirius system classes 113
- Sirius Wiki, menu option 38, 379
- Size of preview 85
- Skip Previewed Source, menu option 29, 86
- Skip Whole Programs 13, 75, 81
- skipPreview command 86, 268
- Socket object, Sirius 113
- SOCKUSER thread 345, 348
- SortedRecordset object, Sirius 113
- Source Code tab 11
- source code, preview of 18, 85
- Source Preview feature 85
- Source Preview, Preferences option 18
- span command 260, 269
- SSL parameter, JANUS DEFINE 368
- SSL ports, Web Server 159, 374, 387
- startup attribute, debuggerConfig.xml file 309, 380
- startUpMacro attribute, ui.xml file 287, 305, 309

stateFileFolder element, configuration file 296, 378
statement line number 11
statement, executable 54, 55
status bar 50
status messages 50
STATUS option, SIRIUS DEBUG command 151, 152, 156
status report, worker thread 151, 156
StatusMessage method, DebuggerTools class 166
Step button 55
step command 55, 269, 291
Step Next button 40
step out operation 64
Step Out, menu option 29, 64
Step Over button 40, 56, 162
Step Over, menu option 29, 56
Step, menu option 29, 55
stepOut command 270, 291
stepOver command 56, 270, 291
stopOnAssertFailure option, setPreference command 260
string &&functions 322
Stringlist object variables 98, 104, 113, 205
StringTokenizer object, Sirius 113
structures, Janus SOAP 87
SUSPEND option, SIRIUS DEBUG command 151, 155, 156
suspend-daemon-debugging mode 142
system exceptions 111

T

text color 18, 295
Text color panel 295
text copying, printing, or saving 149
TEXT statement, User Language 96, 295
Text Viewer window 149
textviewer.xml file 297
The page cannot be displayed 357
tilde character (~) 44
Toggle Breakpoint on Current Line, menu option 27, 58, 60
Toggle Breakpoint option 58, 60, 61
toggle command 271
Toggle Init Exclude, menu option 29, 74
toggleBreakpointOnCurrentLine command 58, 60, 188, 272
toggleInitExclude command 74, 273
toggleLower command 273, 305

Top button 40, 301
top command 274, 291
Top option, Preferences window 41
Top, menu option 24
Trace All button 48
Trace button 48, 130, 132
trace command 274, 291
Trace To End, menu option 29, 130
Trace Until Variable Equals Value, menu option 29, 134
Trace Values, menu option 29, 132
traceUntilVariableEqualsValue command 134, 275
traceValues command 132, 275
Tracing dialog box 134
tracing execution 129
 all lines 29, 130
 of variable value updates 29, 132
 until a specified value 29, 134
 until value changes 29, 133
tracing, Debugger Server port 345, 348
tracing, simple 130
Trim blanks 18, 149
Trim blanks from selection in View Text, Preferences option 18
troubleshooting 345, 348
tuning, Client performance 359
Turn Off Black List button 276
Turn Off Black List, menu option 29, 81, 276
Turn Off Debugging, menu option 29
Turn off Lists button 81
Turn Off White List, menu option 29, 81
Turn On Black List button 81, 277
Turn On Black List, menu option 29, 81
Turn on White List button 81
Turn On White List, menu option 29, 81
turnOffBlackList command 29, 81, 276
turnOffDebugging command 7, 154, 276
turnOffWhiteList command 29, 81, 276
turnOnBlackList command 29, 81, 277
turnOnWhiteList command 29, 81, 277

U

ui.xml file 43, 149, 287, 291, 297, 314
uiFolder element, configuration file 296, 378
uimore.xml file 291, 297, 314
UL folder 371
UltraEdit editor 166, 172
Unicode %variables 124
Universal Buffer, Model 204 123

UnknownStatistic object, Sirius 113
 unPin command 278, 303
 UnPin option, context menu 278, 303
 unSet command 279
 until.txt file 75, 297
 unzip.exe file 371
 updateGet.exe file 38, 371, 390
 Updates, menu option 38, 390
 Use !debugger directives checkbox 260
 Use !debugger directives, Preferences option 18, 68, 260
 Use automatic configuration script, IE option 209, 258, 383, 385
 Use automatic configuration script, Internet Explorer 260, 383
 Use existing IE proxy for URLs not to be debugged, Preferences option 19, 361, 385
 Use Proc Lists for exclude/include 70, 239, 260
 Use Routine Lists for exclude/include 70, 239, 260
 useDefaults attribute, ui.xml file 287
 useProcLists option, setPreference command 260
 User Language
 watchable entities 87
 useRoutineLists option, setPreference command 260
 UserStatistics object, Sirius 113

V

value
 details of 33, 93, 101, 104, 106, 111, 116, 208
 Value button 33, 49, 93, 101, 279
 Value Display, menu option 33
 Value window 33, 87, 93, 101, 343
 valueDisplay command 101, 279
 valueDisplay.xml file 297
 valueDisplayOnConsole option, setPreference command 101, 260, 279, 316
 varDump command 280, 321
 variable value
 setting 124
 tracing until change 133
 tracing updates to 132
 updating 6
 variables 87
 array 87
 class 100, 111, 211

Debugger macro 271, 280, 314, 319
 global 96
 object 97
 watched 87
 vars.txt file 297
 View PDF Manual, menu option 7, 38
 View Text button 149
 View Text, menu option 24, 149
 viewText command 10, 149, 281

W

Waiting for Online message 354, 358, 388, 425
 waiting messages 50, 354, 358
 Watch button 33, 49, 88
 Watch Window box
 add or remove items 33, 88
 clearing 40
 contents 18, 334, 336
 description 15, 87
 external window 300
 long items 18, 90
 remembered items 91
 resizing 88
 saving/restoring contents 91, 220, 246
 Watch Window window 300
 watched entity 87, 93, 334, 336
 watchmemory.txt file 297
 Web Browser text box 63, 144, 146
 Web Browser, Preferences option 18
 Web Buffer tab 12, 129
 capacity of 6
 web output buffer 129
 Web request will NOT be debugged, message 374
 Web Server Selection list 147
 Web Server Selection, Preferences option 18
 Web Server, Janus
 debugging multiple instances of 147, 374
 debugging with Sirius Debugger 157
 port number 374
 Web Service application 160
 webPort element, configuration file 374
 White list is active, message 81
 White List processing 13, 29, 79, 162
 White list reloaded message 82
 whitelist.txt file 79, 238, 297, 371
 wiki, Sirius 379
 wikiURL element, configuration file 379
 wildcards, procedure name 70, 75, 76, 79

Index

Window menu 24
window, external 24, 193, 194, 195, 231, 232,
233, 236, 278, 281, 300, 303, 343
window, main 14, 243, 264, 330, 337
windowmemory.xml file 297
Windows 7 406
Windows When Suspended option 63, 144
Windows When Suspended, Preferences option
18
Windows, supported versions of 6
windowToTop command 281
work files, Debugger Client 296, 378
worker port 374
worker thread lost 354
worker thread, Debugger Server 4, 151, 156,
345, 348, 366, 367
workerPort element, configuration file 374
workstation, Debugger Client
host ID 370
port number 370, 376
Wrap button 93, 101, 123

X

XML document variables, debugging 106
XmlDoc object, Sirius 113
XmlNode object, Sirius 113
XmlNodeList object, Sirius 113
Xtend editor 166, 168
XTEND subsystem 168