



Custom Server Agent

Technical Note

UPDATED: 27 July 2023

Copyright Notices

© 2022 Progress Software Corporation and/or one of its subsidiaries or affiliates. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

#1 Load Balancer in Price/Performance, 360 Central, 360 Vision, Chef, Chef (and design), Chef Habitat, Chef Infra, Code Can (and design), Compliance at Velocity, Corticon, Corticon.js, DataDirect (and design), DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Defrag This, Deliver More Than Expected, DevReach (and design), Driving Network Visibility, Flowmon, Inspec, Ipswitch, iMacros, K (stylized), Kemp, Kemp (and design), Kendo UI, Kinvey, LoadMaster, MessageWay, MOVEit, NativeChat, OpenEdge, Powered by Chef, Powered by Progress, Progress, Progress Software Developers Network, SequeLink, Sitefinity (and Design), Sitefinity, Sitefinity (and design), Sitefinity Insight, SpeedScript, Stylized Design (Arrow/3D Box logo), Stylized Design (C Chef logo), Stylized Design of Samurai, TeamPulse, Telerik, Telerik (and design), Test Studio, WebSpeed, WhatsConfigured, WhatsConnected, WhatsUp, and WS_FTP are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries.

Analytics360, AppServer, BusinessEdge, Chef Automate, Chef Compliance, Chef Desktop, Chef Workstation, Corticon Rules, Data Access, DataDirect Autonomous REST Connector, DataDirect Spy, DevCraft, Fiddler, Fiddler Classic, Fiddler Everywhere, Fiddler Jam, FiddlerCap, FiddlerCore, FiddlerScript, Hybrid Data Pipeline, iMail, InstaRelinker, JustAssembly, JustDecompile, JustMock, KendoReact, OpenAccess, PASOE, Pro2, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Push Jobs, SafeSpaceVR, Sitefinity Cloud, Sitefinity CMS, Sitefinity Digital Experience Cloud, Sitefinity Feather, Sitefinity Thunder, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Supermarket, SupportLink, Unite UX, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

Please refer to the NOTICE.txt or Release Notes – Third-Party Acknowledgements file applicable to a particular Progress product/hosted service offering release for any related required third-party acknowledgements.

Table of Contents

1 How to Write a Custom Server Agent	4
Last Updated Date	11

1 How to Write a Custom Server Agent

This document outlines how to create your own adaptive load balancing server agent for the LoadMaster. It focuses on the requirements for creating and configuring such agents on arbitrary server operating systems – for example, it describes the protocol used by the agent to communicate with the LoadMaster. It does not focus on the mechanics of generating a load value using specific Operating System (OS) utilities, although a simple example of doing this is presented for Linux systems.

For Real Servers running Windows, you can use the Windows server agent supplied by Kemp, or write your own PowerShell script that uses Windows commands equivalent to the ones used in the examples in this section.

When adaptive load balancing is selected for a Virtual Service (VS), the LoadMaster expects that there is a program, called a ‘server agent’, running on each Real Server in the VS. The LoadMaster expects that the server agent performs the following tasks:

1. Periodically queries the Real Server operating system for whatever system load statistics are available and desired.
2. Uses the load values obtained from the operating system to arrive at an integer value.
3. Places the integer value in an otherwise empty file on the Real Server that can be queried using an HTTP GET command from the LoadMaster.

Point **3** requires that there is an HTTP server running on the Real Server that makes the file containing the integer load value available to the LoadMaster.

The server agent and an appropriately configured HTTP server must be running on all Real Servers in a VS before selecting adaptive load balancing for the VS on the LoadMaster.

On the LoadMaster side:

1. Once the adaptive scheduling method is chosen on a VS, the LoadMaster periodically attempts to retrieve the appropriate file (specified by the user) from each Real Server and read the integer load value from the file.

1 How to Write a Custom Server Agent

2. Until the LoadMaster successfully retrieves an integer load value from a Real Server, it will be considered unavailable.

3. Once an integer load value is retrieved, the LoadMaster modifies the Real Server's status according to the table below.

Agent Response	Meaning and Action Taken by the LoadMaster
No HTTP Response	If the LoadMaster does not get a response when it attempts to get the file containing the load value, then the Real Server is assumed to be unavailable and it is marked down. The server dynamic weight is set to 0 so that no new connections are sent to it; currently open connections are not affected.
File does not exist or file is empty	If the HTTP server is running on the Real Server, but the required file either does not exist or is empty (contains no data), then the server is assumed to be available, it is marked up, and the server dynamic weight is set to 100.
Non-integer value	If the file returned by the HTTP server running on the Real Server contains a non-integer value, then the server is assumed to be available, it is marked up, and the server dynamic weight is set to 100.
< 0	If the file returned by the HTTP server running on the Real Server contains a negative integer value (for example, -1), then the server is assumed to be available, it is marked up, and the server dynamic weight is set to 100.
0	If the file returned by the HTTP server running on the Real Server contains an integer value of 0 (zero), this forces the LoadMaster to temporarily fall back to using the round robin scheduling method until the server agent returns a value other than 0. This return value is intended to be used in a situation where the agent code cannot for some reason arrive at a valid integer value to return to the LoadMaster. This might happen if, for example, the server is not responding to the agent's queries for load values.
1-100	If the file returned by the HTTP server running on the Real Server contains an integer value between 1 and 100, the server is considered to be loaded to the indicated level, which is a percentage of fully loaded. The LoadMaster sets the dynamic weight appropriately: the higher the number returned by the server agent, the lower the dynamic weight of the server will be set.
101	If the file returned by the HTTP server running on the Real Server contains an integer value of 101, the server is considered to have reached its maximum load and it is marked down. The server dynamic weight is set to 0 so that no new connections are sent to it; currently open connections are not affected.

102	If the file returned by the HTTP server running on the Real Server contains an integer value of 102, this means essentially the same as returning 101 except that the LoadMaster also drops all currently open connections if the Drop connections on RS failure option is set on the associated VS.
> 102	Treated as if the server returned 101.

On Linux systems, a straightforward manner in which to implement a server agent is to use a cron script to generate a load value and place it in the appropriate location on a regular basis – for example, every 5 seconds.

Such a script could be as simple as this:

```
#!/bin/bash
top -l1 | awk '{printf "%d\n", $7}' > /path/load
```

The above uses the top command and a short awk script to copy the current CPU load value in a file – the path above specifies the path to the root of an HTTP server on the Real Server. The filename ‘load’ is the default filename retrieved by the LoadMaster. [This and other adaptive load balancing options are located in the User Interface (UI) under **Rules & Checking > Check Parameters > Adaptive Parameters.**]

Alternatively, you could instead write a CGI program (like the Kemp-supplied Windows server agent) or any other program that the HTTP server can be configured to run when the LoadMaster sends a GET for the required file to the HTTP server. This program would generate the integer load value when the LoadMaster requests the file from the server and the HTTP server would return it to the LoadMaster.

There are advantages and disadvantages to either approach. For example:

- Using a static file populated using a cron job is a simple and easy-to-maintain option, but also means that the data in the file could be stale when it is accessed by the LoadMaster.
- Using a program that runs each time the LoadMaster checks performance means that you will get the most accurate data, but the server agent itself could contribute to system load if CPU, Disk I/O, and network performance were all being continually checked by the server agent.

The approach you choose will most likely depend on the needs of your configuration and the tools you typically use.

In terms of internal processing, a server agent can be as complicated as you like, depending on your application. If your application is completely CPU-intensive, then a simple script like the two-line bash script above may be enough. But, if your application also generates significant disk I/O or

1 How to Write a Custom Server Agent

network-traffic, you will probably want to consider the load on those resources as well as CPU usage to get a more detailed view of server performance.

The following script is an example of gathering load information on CPU and memory, and combining them into a single integer value that the LoadMaster can use for adaptive load balancing. It is written in `bash` shell and is intended to be run on a Linux server using a privileged `crontab` file. See the Linux manual pages for `bash`, `crontab`, and the commands used in the script to gather the required statistics. See the comments in the script for details.

This script is intended solely as an example, although it could be used in a demonstration or proof of concept of how to configure a LoadMaster and a Real Server for adaptive load balancing.

```
#!/bin/bash
#
# Very simple adaptive server agent program, intended to be run using a root
# or similarly privileged crontab(1) entry to periodically populate
# the file the LoadMaster expects to be available using HTTP on the Real
# Server.
##### DEFAULTS
# This is the default filename expected by the LoadMaster. This setting must
# match what is on "Rules & Checking > Adaptive Parameters" in the LoadMaster
# Web User Interface (WUI).
LOAD_FILE_PATH=/load
# This is the default doc root of Apache2 on an Ubuntu server, and where
# we will place the 'load' file defined above for the LoadMaster to GET.
DOCR00T=/var/www/html
# Define the weights to be used for CPU and Memory.
# Change these numbers to indicate the percentage weight that each
# statistic should carry when figuring the overall load value to
# return to the LoadMaster. By default, these are set to 0.25 for CPU and
0.75
# for Memory, so the effect of the weighting is apparent. Weights must
# add up to 1.
CPUweight=0.25
MEMweight=0.75
##### GET MEMORY STATS
# Get memory usage from the free(1) command using a simple awk command and
```

```
# calculate the percentage of memory used. Since the shell does integer
# arithmetic only, we will use bc(1) to do the math.
MEMtotal=`free | awk '/Mem:/ {print $2}'`
MEMstatus=$?
MEMused=`free | awk '/Mem:/ {print $3}'`
MEMload=`echo "scale=3;( $MEMused / $MEMtotal ) * 100" | bc`
# Round the answer to the nearest integer
MEMloadrnd=`echo "$MEMload + 0.5" | bc`
MEMloadrnd=`echo "scale=0; $MEMloadrnd/1" | bc`
#### GET CPU LOAD
# Note: Some version of top(1) report avg. CPU usage since time of last
# reboot in the first iteration, so we will take the second iteration value.
CPUidletop=`top -b -n2 -d.2 | awk '/%Cpu/ {print $8}'`
CPUidle=`echo $CPUidletop | cut -d" " -f2`
CPUload=`echo "scale=3;100 - $CPUidle" | bc`
CPUstatus=$?
# Round the answer to the nearest integer
CPUloadrnd=`echo "$CPUload + 0.5" | bc`
CPUloadrnd=`echo "scale=0; $CPUloadrnd/1" | bc`
#### CHECK IF STATS ARE A VALID % (between 1 and 100)
# VALID_STATS=1 means stats are valid.
# if stats are not valid, we want to return 0 to LM to disable adaptive
# LB and use round robin until a different value is received.
VALID_STATS=1
if [ $MEMloadrnd -lt 1 -o $MEMloadrnd -gt 100 -o $CPUloadrnd -lt 1 \
-o $CPUloadrnd -gt 100 ]
then
VALID_STATS=0
RETURNED_LOAD=0
fi
#### CALCULATE THE WEIGHTED AVERAGE OF $CPUload AND $MEMload.
```



```
# Only do this if stats were valid - that is, $VALID_STATS=1.
if [ $VALID_STATS -eq 1 ]
then
RSload=`echo "( $CPUweight * $CPUloadrnd ) + \
( $MEMweight * $MEMloadrnd )" | bc`
RSloadrnd=`echo "$RSload + 0.5" | bc`
RSloadrnd=`echo "scale=0; $RSloadrnd/1" | bc`
RETURNED_LOAD=$RSloadrnd
fi

#### DETERMINE IF WE SHOULD RETURN 101 OR 102.
# Returning 101 or 102 in the case of a highly loaded system is somewhat
# a matter of policy. In this example, we will return 101 (take the server
# out of rotation) when stats are valid AND the aggregated load value
# is above 90%, to allow more resources to become available. If the
# aggregate load goes above 96%, we will return 102 (which takes the server
# out of rotation AND drops all current connections). Presumably, once
# more resources are available, load values will decrease, and the
# server agent will return a lower value on a subsequent run.
if [ $VALID_STATS -eq 1 -a $RSloadrnd -gt 90 ]
then
RETURNED_LOAD=101
elif [ $VALID_STATS -eq 1 -a $RSloadrnd -gt 96 ]
then
RETURNED_LOAD=102
fi

#### PUT THE LOAD VALUE IN THE FILE.
# If stats were not valid above, then return 0 -- this disables adaptive
# load balancing and uses round robin instead; else, return the rounded
# load.
echo $RETURNED_LOAD > ${DOCRROOT}${LOAD_FILE_PATH}

#### PRINT EVERYTHING TO STDOUT FOR DEBUGGING
```

```
# This allows for running the script from the command line to test in
# your environment -- or, if the script is run using cron, cron
# captures the output and emails it to you so you can debug any issues.
# Once you are happy the script is working properly, comment these lines
# out to avoid being emailed output by cron every time the script runs.
echo RETURNED_LOAD = $RETURNED_LOAD
echo RSload = $RSload
echo RSloadrnd = $RSloadrnd
echo MEMweight = $MEMweight
echo MEMtotal = $MEMtotal
echo MEMused = $MEMused
echo MEMload = $MEMload
echo MEMloadrnd = $MEMloadrnd
echo MEMstatus = $MEMstatus
echo CPUweight = $CPUweight
echo CPUidle = $CPUidle
echo CPUload = $CPUload
echo CPUloadrnd = $CPUloadrnd
echo CPUstatus = $CPUstatus
#### EXIT
if [ $VALID_STATS -eq 1 -a $MEMstatus -eq 0 -a $CPUstatus -eq 0 ]
then
# success
exit 0
else
# error
exit 1
fi
#### EOF
```

Last Updated Date

This document was last updated on 27 July 2023.