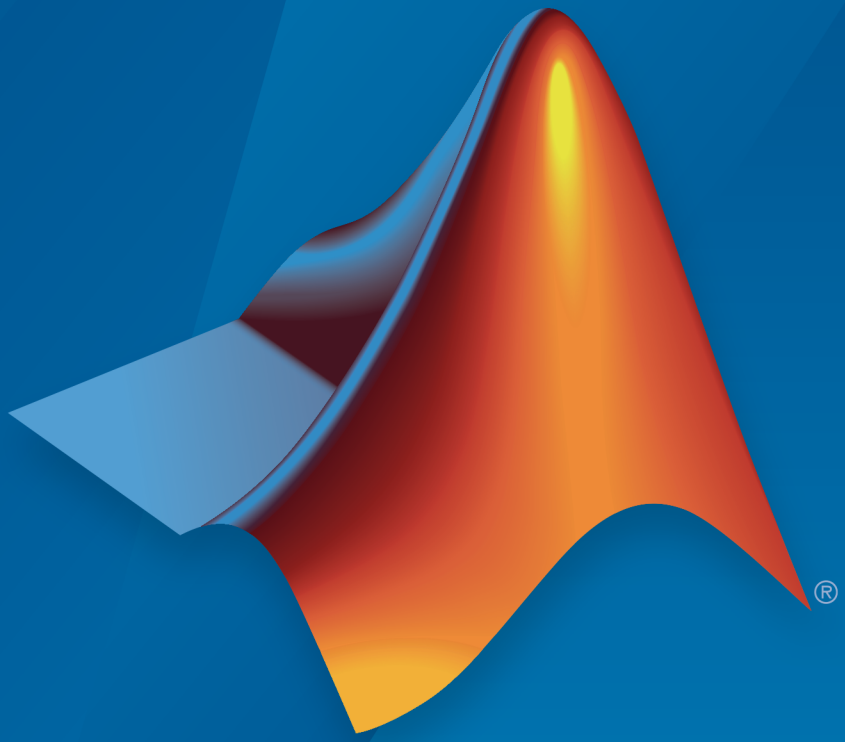
MATLAB® Production Server™

Server Management Guide



MATLAB®

R2016a

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

MATLAB® Production Server™ Management Guide

Revision History

March 2014	Online only	New for Version 1.2 (Release R2014a)
October 2014	Online only	Revised for Version 2.0 (Release R2014b)
March 2015	Online only	Revised for Version 2.1 (Release R2015a)
September 2015	Online only	Revised for Version 2.2 (Release R2015b)
March 2016	Online only	Revised for Version 2.3 (Release 2016a)

Contents

1 Server Management

2 Manage Licenses for MATLAB Production Server

3 Secure a Server

4 Troubleshooting

Server Management

Server Overview

In this section...

What Is a Server?

You can create any number of server instances using MATLAB Production Server software. Each server instance can host any number of deployable archives containing MATLAB code. You may find it helpful to create one server for all archives relating to a particular application. You can also create one server to host code strictly for testing, and so on.

A *server instance* is considered to be one unique *configuration* of the MATLAB Production Server product. Each configuration has its own options file (`main_config`) and diagnostic files (`log` files, Process Identification (`pid`) files, and `endpoint` files).

In addition, each server has its own `auto_deploy` folder, which contains the deployable archives you want the server to host for clients.

The server also manages the MATLAB Runtime, which enables MATLAB code to execute. The settings in `main_config` determine how each server interacts with the MATLAB Runtime to process clients requests. You can set these parameters according to your performance requirements and other variables in your IT environment.

How Does a Server Manage Work?

A server processes a transaction using these steps:

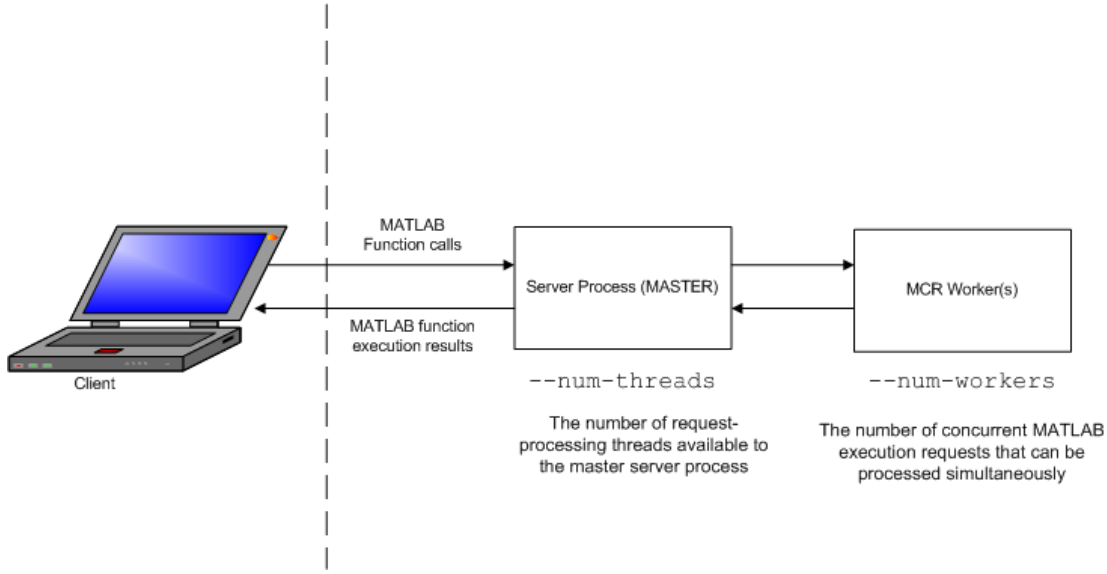
1. The client sends MATLAB function calls to the master server process (the main process on the server).
2. MATLAB function calls are passed to one or more *MATLAB Runtime workers*.
3. MATLAB functions are executed by the MATLAB Runtime worker.
4. Results of MATLAB function execution are passed back to the master server process.
5. Results of MATLAB function execution are passed back for processing by the client.

The server is the middleman in the MATLAB Production Server environment. It simultaneously accepts connections from clients, and then dispatches MATLAB Runtime

workers—MATLAB sessions—to process client requests to the MATLAB Runtime. By defining and adjusting the number of workers and threads available to a server, you tune capacity and throughput respectively.

- Workers (capacity management) (`num-workers`) — The number of MATLAB Runtime workers available to a server.

  Each worker dispatches one MATLAB execution request to the MATLAB Runtime, interacting with one client at a time. By defining and tuning the number of workers available to a server, you set the number of concurrent MATLAB execution requests that can be processed simultaneously. `num-workers` should roughly correspond to the number of cores available on the local host.
- Threads (throughput management) (`num-threads`) — The number of threads (units of processing) available to the master server process.



MATLAB Production Server Data Flow from Client to Server and Back

The server does not allocate a unique thread to each client connection. Rather, when data is available on a connection, the required processing is scheduled on a *pool* of threads. `--num-threads` sets the size of that pool (the number of available request-processing threads) in the master server process. The threads in the pool

do not execute MATLAB code directly. Instead, there is a single thread within each MATLAB Runtime worker process that executes MATLAB code on the client's behalf. The number of threads you define to a server should roughly correspond to the number of cores available on the local host.

Create a Server

In this section...

Prerequisites

Before creating a server, ensure you have:

- Installed MATLAB Production Server software.
- Added the `script` folder to your system `PATH` environment variable. Doing so enables you to run server commands such as `mps-new` from any folder on your system.

Note: You can run server commands from the `script` folder. The script folder is located at *`$MPS_INSTALL`*`\script`, where *`$MPS_INSTALL`* is the location where MATLAB Production Server is installed. For example, on Windows, the default location is: `C:\Program Files\MATLAB\MATLAB Production Server \`*`ver`*`\script`. *`ver`* is the version of MATLAB Production Server.

Procedure

Before you can deploy your MATLAB code with MATLAB Production Server, you need to create a server to host your deployable archive.

A server instance is considered to be one unique *configuration* of the MATLAB Production Server product. Each configuration has its own parameter settings file (`main_config`) as well as its own set of diagnostic files.

To create a server configuration or *instance*, do the following:

1. From the system command prompt, navigate to where you want to create your server instance.
2. Enter the `mps-new` command from the system prompt:

   ```
   mps-new [path/]server_name [-v]
   ```

   where:

- *path* is the path to the server instance and configuration you want to create for use with the MATLAB Production Server product. When specifying a path, ensure the path ends with the *server_name*.

  If you are creating a server instance in the current folder, you do not need to specify a full path. Only specify the server name.
- *server_name* — is the name of the server instance and configuration you want to create.
- `-v` — enables verbose output, giving you information and status about each folder created in the server configuration.

Upon successful completion of the command, MATLAB Production Server creates a new server instance.

Edit the Configuration File

In this section...

About the Server Configuration File

To change any MATLAB Production Server properties, edit the `main_config` configuration file that corresponds to your specific server instance:

`server_name``/config/main_config`

When editing `main_config`, remember these coding considerations:

- Each server has its own `main_config` configuration file.
- You enter only one configuration property and related options per line. Each configuration property entry starts with two dashes (`--`).
- Any line beginning with a pound sign (#) is ignored as a comment.
- Lines of white space are ignored.

Common Customizations

Setting Default Port Number for Client Requests

Use the `http` property to set the default port number on which the server listens for client requests.

Setting Number of Available Workers

Use the `num-workers` property to set the number of concurrent MATLAB execution requests that can be processed simultaneously.

Setting Number of Available Threads

Use the `num-threads` property to set the number of request-processing threads available to the master server process.

Note: For .NET Clients, the HTTP 1.1 protocol restricts the maximum number of concurrent connections between a client and a server to two.

This restriction only applies when the client and server are connected remotely. A local client/server connection has no such restriction.

To specify a higher number of connections than two for remote connection, use the NET classes `System.Net.ServicePoint` and `System.Net.ServicePointManager` to modify maximum concurrent connections.

For example, to specify four concurrent connections, code the following:

```
ServicePointManager.DefaultConnectionLimit = 4;
MWClient client = new MWHttpClient(new MyConfig());
MPSClient mpsExample = client.CreateProxy(
            new Uri("http://user01:9910/mpsexample"));
```

Specify the Default MATLAB Runtime for New Server Instances

Each server that you create with MATLAB Production Server has its own configuration file that defines various server management criteria.

The `mps-setup` command line wizard searches for MATLAB Runtime instances and sets the default path to the MATLAB Runtime for all server instances you create with the product.

To run the command line wizard, do the following after first downloading and performing the "Download and Install the MATLAB Runtime":

1. Ensure you have logged on with `administrator` privileges.
2. At the system command prompt, run `mps-setup` from the `script` folder.

   Alternately, add the `script` folder to your system `PATH` environment variable to run `mps-setup` from any folder on your system. The `script` folder is located at *`$MPS_INSTALL`*`\script`, where *`$MPS_INSTALL`* is the location in which MATLAB Production Server is installed. For example, on Windows®, the default location is `C:\Program Files\MATLAB\MATLAB Production Server\`*`ver`*`\script\mps-setup`.

   `ver` is the version of MATLAB Production Server to use.
3. Follow the instructions in the command line wizard.

   The wizard will search your system and display installed MATLAB Runtime instances.
4. Enter `y` to confirm or `n` to specify a default MATLAB Runtime for all server configurations created with MATLAB Production Server.

   If `mps-setup` cannot locate an installed MATLAB Runtime on your system, you will be prompted to enter a path name to a valid instance.

Run mps-setup in Non-Interactive Mode for Silent Install

You can also run `mps-setup` without interactive command input for silent installations.

To run `mps-setup`, specify the path name of the MATLAB Runtime as a command line argument. For example, on Windows:

```
mps-setup "C:\Program Files\MATLAB\MATLAB Runtime\mcrver"
```

mcrver is the version of the MATLAB Runtime to use.

Specify the MATLAB Runtime for a Server Instance

To specify the installed location of the MATLAB Runtime for your server instance:

1. If the server instance is running, stop it.
2. Open the configuration file for the instance in a text editor.

   The configuration file is located at *`instanceRoot`*`/config/main_config`.
3. Locate the entry for the `mcr_root` property.

   ```
   --mcr-root mCRuNsETtOKEN
   ```
4. Modify the `mcr_root` property to point to the installed MATLAB Runtime you want to work with.

   For example:

   ```
   --mcr-root C:\Program Files\MATLAB\MATLAB Runtime\vnnn
   ```

   Note: You *must* specify the version number of the MATLAB Runtime (v*nnn*). MATLAB Runtime versions you specify must be compatible with MATLAB Production Server.
5. Restart the server instance.

Start a Server Instance

In this section...

Prerequisites

Before attempting to start a server, verify that you have:

- Installed the MATLAB Runtime
- Created a server instance
- Specified the default MATLAB Runtime for the instance

Procedure

To start a server instance, complete the following steps:

1. Open a system command prompt.
2. Enter the `mps-start` command:

   ```
   mps-start [-C path/]server_name [-f]
   ```

   where:

   - `-C` *`path`*`/` — Path to the server instance you want to create. *`path`* should end with the server name.
   - *`server_name`* — Name of the server instance you want to start or stop.
   - `-f` — Forces command to succeed, regardless or whether the server is already started or stopped.

   Upon successful completion of the command, the server instance is active.

Note: If needed, use the `mps-status` command to verify the server is running.

Share the Deployable Archive

After you create the deployable archive, share it with clients of MATLAB Production Server by copying it to your server, for hosting.

In order to share the deployable archive, a server must be created and started.

1. Locate your deployable archive in the `for_redistribution_files_only` folder of your compiler project folder.

   It is named *`project_name`*`.ctf`.

2. Copy *`project_name`*`.ctf` to the `\`*`server_name`*`\auto_deploy` folder in your server instance.

   For example, if your server is named `prod_server_1` and located in `C:\tmp`, copy *`project_name`*`.ctf` to `C:\tmp\prod_server_1\auto_deploy`.

Support Multiple MATLAB Versions

In this section...

MATLAB Production Server instances can host deployable archives compiled using multiple versions of MATLAB Compiler SDK™. You configure a server instance to do this by adding multiple `mcr-root` properties to the configuration file for the instance:

1. Install the required versions of the MATLAB Runtime.

   Note:

   - A server instance should only be configured to use MATLAB Runtime roots on a local file system. Otherwise, a network partition may cause worker processes to fail.
   - All values for `mcr-root` must be for the same OS/hardware combination.

2. If the server instance is running, stop it.
3. Open the configuration file for the instance in a text editor.

   The configuration file is at *`instanceRoot`*`/config/main_config`.
4. Locate the entry for the `mcr-root` property.

   ```
   --mcr-root mCRuNsETtOKEN
   ```
5. For each version of the MATLAB Runtime the instance supports, add an instance of the `mcr_root` property.

   For example, to configure the instance to use the v81 and v82 versions of the MATLAB Runtime.

   ```
   --mcr-root C:\Program Files\MATLAB\MATLAB Compiler Runtime\v81
   --mcr-root C:\Program Files\MATLAB\MATLAB Runtime\v82
   ```
6. Restart the server instance.

How the Server Instance Selects the MATLAB Runtime to Use

Once the server instance is configured to use multiple versions of MATLAB Runtime, it scans the list of provided MATLAB Runtime installations in order from first to last

and chooses the first MATLAB Runtime installation capable of processing the request. A MATLAB Runtime installation can process a request if it is compatible with the version of MATLAB used to create the deployable archive containing the function being evaluated.

Note: Since the server instance always chooses the first compatible version of MATLAB Runtime, configuring the server instance with multiple instances of the same MATLAB Runtime version has no effect on performance.

Changes to Worker Management

Configuring a server instance to use multiple MATLAB Runtime versions also changes how the server instance manages the workers used to process requests.

When using a single MATLAB Runtime installation, the server instance starts workers as needed until `num-workers` workers are running. Once running, workers maybe restarted in response to the `worker-restart-interval` property or the `worker-restart-memory-limit` property. Workers are never fully stopped.

Once a server instance starts using multiple MATLAB Runtime versions, it dynamically manages the worker pool. The server instance starts new workers as needed until `num-workers` workers are running. The worker instances are spread out over the different MATLAB Runtime versions. Once `num-workers` workers are running, the server instance returns workers to the pool of available workers based on the `worker-memory-trigger` property and the `queue-time-trigger` property. Once worker is returned to the pool, it can be allocated to process new requests using any of the configured MATLAB Runtime versions.

Control Worker Restarts

In this section...

Restart Workers Based on Up Time

As worker processes evaluate MATLAB functions, the MATLAB workspace accumulates saved state and other data. This accumulated data can occasionally cause a worker process to fail. One way to avoid random worker failures is to configure the server instances to restart worker processes when they have been running for set period.

1. If the server instance is running, stop it.
2. Open the configuration file for the instance in a text editor.

   The configuration file is at *`instanceRoot`*`/config/main_config`.
3. Locate the entry for the `worker-restart-interval` property.

   ```
   --worker-restart-interval 12:00:00
   ```
4. Change the value to the desired restart interval.

   For example, restart workers at intervals of 1 hour, 29 minutes, 5 seconds.

   ```
   --worker-restart-interval 1:29:05
   ```
5. Restart the server instance.

Restart Workers Based on Amount of Memory in Use

As worker processes evaluate MATLAB functions, the MATLAB workspace accumulates saved state and other data. This accumulated data can occasionally cause a worker process to fail. One way to avoid random worker failures is to configure the server instances to restart worker processes when they begin consuming a predefined amount of memory.

This is done by adjusting three configuration properties:

- `worker-memory-check-interval` — Interval at which workers are polled for memory usage

- `worker-restart-memory-limit` — Size threshold at which to consider restarting a worker
- `worker-restart-memory-limit-interval` — Interval for which a worker can exceed its memory limit before restart

To adjust memory-based restart thresholds:

1. If the server instance is running, stop it.
2. Open the configuration file for the instance in a text editor.

   The configuration file is at *`instanceRoot`*`/config/main_config`.
3. Locate the entry for the `worker-memory-check-interval` property.

   ```
   --worker-memory-check-interval 0:00:30
   ```
4. Change the value to the desired restart interval.

   For example, restart workers at intervals of 1 hour, 29 minutes, 5 seconds.

   ```
   --worker-memory-check-interval 1:29:05
   ```
5. Add an entry for the `worker-restart-memory-limit` property.

   For example, consider restarting workers when they consume 1 GB of memory.

   ```
   --worker-restart-memory-limit 1GB
   ```
6. Add an entry for the `worker-restart-memory-limit-interval` property.

   For example, restart workers when they exceed the memory limit for 1 hour.

   ```
   worker-restart-memory-limit-interval 1:00:00
   ```
7. Restart the server instance.

Install a Server Instance as a Windows Service

In this section...

Create a New Server Instance as a Windows Service

To create a new MATLAB Production Server instance and register it as a Windows service, use the `mps-new` command with the `--service` option.

```
mps-new /tmp/server_1 --service
```

You can change the name, description, and user for the Windows service from the defaults using optional flags to the `mps-new` command.

The Windows service created for the server instance does not start automatically. You can edit the configuration for the instance before starting it is using the `mps-start` command.

The Windows service created for the server instance is configured to start when the machine starts. When the host machine is restarted, the server instance restarts with it.

Make an Existing Server Instance a Windows Service

To create a new Windows service for an existing MATLAB Production Server instance, use the `mps-service` command with the `create` option.

```
mps-service -C /tmp/server_1 create
```

You can change the name, description, and user for the Windows service from the defaults using optional flags to the `mps-service` command.

The Windows service created for the server instance is configured to start when the machine starts. When the host machine is restarted, the server instance restarts with it.

2

Manage Licenses for MATLAB Production Server

Specify or Verify License Server Options in Server Configuration File

Specify or verify values for License Server options in the server configuration file (`main_config`). You create a server by using the `mps-new` command.

Edit the configuration file for the server. Open the file *`server_name`*`/config/main_config` and specify or verify parameter values for the following options. See the comments in the server configuration file for complete instructions and default values.

- `license` — Configuration option to specify the license servers and/or the license files. You can specify multiple license servers including port numbers (*`port_number`*`@`*`license_server_name`*), as well as license files, with one entry in `main_config`. List where you want the product to search, in order of precedence, using semi-colons (;) as separators on Windows or colons (:) as separators on Linux.

  For example, on a Linux system, you specify this value for `license`:

  ```
  --license 27000@hostA:/opt/license/license.dat:27001@hostB:./license.dat
  ```

  The system searches these resources in this order:

  1. `27000@hostA:` (`hostA` configured on port `27000`)
  2. `/opt/license/license.dat` (local license data file)
  3. `27001@hostB:` (`hostB` configured on port `27001`)
  4. `./license.dat` (local license data file)
- `license-grace-period` — The maximum length of time MATLAB Production Server responds to HTTP requests, after license server heartbeat has been lost. See FLEXlm® documentation for more on heartbeats and related license terminology.
- `license-poll-interval` — The interval of time that must pass, after license server heartbeat has been lost and MATLAB Production Server stops responding to HTTP requests, before license server is polled, to verify and checkout a valid license. Polling occurs at the interval specified by `license-poll-interval` until license has been successfully checked-out. See FLEXlm documentation for more on heartbeats and related license terminology.

Verify Status of License Server using mps-status

When you enter an `mps-status` command, the status of the server *and* the associated license is returned.

For detailed descriptions of these status messages, see "License Server Status Information".

Force a License Checkout Using mps-license-reset

Use the `mps-license-reset`command to force MATLAB Production Server to checkout a license. You can use this command at any time, providing you do not want to wait for MATLAB Production Server to verify and checkout a license at an interval established by a server configuration option such as `license-grace-period` or `license-poll-interval`.

3

Secure a Server

Overview

MATLAB Production Server uses HTTPS to establish secure connections between server instances and clients. The HTTPS layer provides certificate-based authentication for both clients and server instances. It also provides an encrypted data path between the clients and server instances. You can configure the level of security provided by the HTTPS layer and the security protocols it supports.

MATLAB Production Server provides a certificate-based authorization mechanism for restricting access to specific programs. Using this mechanism, you specify the MATLAB programs that a client can access.

Enable Security

To enable security, add the following to the server instance's configuration:

- HTTPS port
- Valid certificate stored in a PEM formatted certificate chain
- Valid private key stored in PEM format

The following configuration excerpt configures a server instance to accept secure connections on port 9920, use the certificate stored in `./x509/my-cert.pem`, and use the unencrypted private key stored in `./x509/my-key.pem`.

```
...
--https 9920
--x509-cert-chain ./x509/my-cert.pem
--x509-private-key ./x509/my-key.pem
...
```

The default security settings allow all clients to access all programs hosted by the server instance. The server instance does not authenticate the clients, nor does it perform any authorization. The default settings enable all security protocols and enable all cipher suites, save for eNULL.

In production settings that require greater security than that provided by an unencrypted private key, use an encrypted private key. You specify the passphrase for decrypting the private key in a file with owner-read-only access, and use the `--x509-passphrase` property to tell the server instance about it.

```
...
--https 9920
--x509-cert-chain ./x509/my-cert.pem
--x509-private-key ./x509/my-key.pem
--x509-passphrase ./x509/my-passphrase
...
```

Configure Client Authentication

To ensure that only trusted client applications have access to a server instance, configure the server instance to require client authentication:

1. Set the `ssl-verify-peer-mode` configuration property to `verify-peer-require-peer-cert`.
2. Configure the server instance to use the system provided CA store, a server specific CA store, or both.

   Use these configuration properties to control the CA stores used by the server instance:

   - `x509-ca-file-store` specifies a PEM formatted CA store to authenticate clients.
   - `x509-use-system-store` directs the server instance to use the system's CA store to authenticate clients.

   Note: `x509-use-system-store` does not work on Windows.
3. Optionally configure the server instance to respect any certificate revocation lists (CRLs) in the CA store.

   Specify this behavior by adding the `x509-use-crl` property to the server's configuration. If this property is not specified, the server instance ignores the CRLs and may authenticate clients using revoked credentials.

   Caution You must add a CRL list to the server's CA store before adding the `x509-use-crl` property. If the CA store does not include a CRL list, the server will crash.

This configuration excerpt configures a server instance to authenticate clients using the system CA store and to respect CRLs:

```
...
--https 9920
--x509-cert-chain ./x509/my-cert.pem
--x509-private-key ./x509/my-key.pem
--x509-passphrase ./x509/my-passphrase
--ssl-verify-peer-mode verify-peer-require-cert
--x509-use-system-store
--x509-use-crl
```

. . .

Specify Access to MATLAB Programs

By default, server instances allow all clients to access all hosted MATLAB programs. You control this behavior using the `ssl-allowed-client` configuration property. The `ssl-allowed-client` property specifies a comma separated list of clients, identified by their certificate's common name, that are allowed to access MATLAB programs. You also use the property to list specific MATLAB programs that a client is allowed to access.

If you do not specify the `ssl-allowed-client` property, the server instance does not restrict access to the hosted MATLAB programs. After you add an entry for the `ssl-allowed-client` property, the server instance authorizes only the listed clients to access the hosted MATLAB programs.

To only authorize clients with the common names `jim`, `judy`, and `ash` to use the MATLAB programs hosted on a server instance, add this configuration excerpt:

```
--ssl-allowed-client jim,judy,ash
```

You can restrict access further by only authorizing specific clients to have access to specific MATLAB programs. Do this by adding `:allowedPrograms` to the value of the `ssl-allowed-client` property. *`allowedPrograms`* is a comma separated list of program names.

To allow clients with the common name `jim` access to all hosted programs, allow clients with the common name `judy` access to the programs `tail` and `zap`, and allow clients with the common name `ash` or `joe` access to the programs `saw` and `travel`, add this configuration excerpt:

```
--ssl-allowed-client jim
--ssl-allowed-client judy:tail,zap
--ssl-allowed-client ash,joe:saw,travel
```

Adjust Security Protocols

By default, MATLAB Production Server instances try to use TLSv1.2 to secure connections between client and server. It will allow connections using SSLv2, SSLv3, and TLSv1. You control the enabled protocols using the `--ssl-protocols` property. This property specifies the protocols the server instance can use.

To disable SSLv2 and SSLv3 protocols, add this configuration excerpt:

```
--ssl-protocols TLSv1
```

Because SSLv2 and SSLv3 are not included in the list, the server instance does not enable the protocols.

You can restrict the cipher suites used by the server instance with the `ssl-ciphers` property. After you add the property to a server instance's configuration, the server instance will use only the listed cipher suites.

To enable only high strength cipher suites, you add this configuration excerpt:

```
--ssl-ciphers HIGH
```

Improve Startup Time When Security Is Activated

When a server instance is configured to use HTTPS, it generates an ephemeral DH key at startup. Generating the DH key at startup provides more security than reading it from a file on disk. However, this can add a couple of minutes to a server instance's startup time.

If you need the server instance to startup without delay and are not concerned about the loss of security, you can configure the server instance to read the ephemeral DH key from a file using the `ssl-tmp-dh-param` configuration property. The `ssl-tmp-dh-param` property specifies the file storing the DH key in PEM format.

4

Troubleshooting

Verify Server Status

In this section...

Procedure

To verify the status of a server instance, complete the following steps:

1. Open a system command prompt.
2. Enter the following command:

   ```
   mps-status [-C path/]server_name
   ```

   where:

   - `-C path/` — Path to the server instance and configuration you want to create. *`path`* should end with the server name.
   - *`server_name`* — Name of the server instance and configuration you want to start or stop.

   Upon successful completion of the command, the server status displays.

License Server Status Information

In addition to the status of the server, `mps-status` also displays the status of the license server associated with the server you are verifying.

Possible statuses and their meanings follow:

This License Server Status Message...	Means...
`License checked out`	The server is operating with a valid license. The server is communicating with the License Manager, and the proper number of license keys are checked out.

This License Server Status Message...	Means...
`WARNING: lost connection to license server - request processing will be disabled at time unless connection to license server is restored`	The server has lost communication with the License Manager, but the server is still fully operational and will remain operational until the specified *`time`*. At *`time`*, if connectivity to the license server has not been restored, request processing will be disabled until licensing is reestablished.
`ERROR: lost connection to license server - request processing disabled`	The server has lost communication with the License Manager for a period of time exceeding the grace period. Request processing has been suspended, but the server actively attempts to reestablish communication with the License Manager until it succeeds, at which time normal request processing resumes.

Verify Status of a Server

This example shows how to verify the status of the server instance you started in the previous example.

In this example, you verify the status of `prod_server_1`, from a location other than the server instance folder (`C:\tmp\prod_server_1`).

1. Open a system command prompt.
2. To verify `prod_server_1` is running, enter this command:

   ```
   mps-status -C \tmp\prod_server_1
   ```

   If `prod_server_1` is running, the following status is displayed:

   ```
   \tmp\prod_server_1 STARTED
   license checked out
   ```

This output confirms `prod_server_1` is running and the server is operating with a valid license.

For more information on the `STOPPED` status, see `mps-stop` and `mps-restart`.

For more information about license status messages, see “License Server Status Information” on page 4-2.

Diagnose a Server Instance

To diagnose a problem with a server instance or configuration of MATLAB Production Server, do the following, as needed:

- Check the logs for warnings, errors, or other informational messages.
- Check Process Identification Files (PID files) for information relating to problems with MATLAB Runtime worker processes.
- Check Endpoint Files for information relating to problems relating to the server's bound external interfaces — for example, a problem connecting a client to a server.
- Use server diagnostic tools, such as `mps-which`, as needed.

Diagnose a Corrupted MATLAB Runtime

This example shows a typical diagnostic procedure you might follow to solve a problem starting server `prod_server_x`.

After you issue the command:

```
mps-start prod_server_x
```

from within the server instance folder (`prod_server_x`), you get the following error:

```
Server process exited with return code: 4
(check logs for more information)
Error while waiting for server to start: The I/O operation
has been aborted because of either a thread exit
or an application request
```

To solve this issue, you might check the `log` files for more detailed messages, as follows:

1. Navigate to the server instance folder (`prod_server_x`) and open the `log` folder.
2. Open `main.err` with any text editor. Note the following message listed under `Server startup error:`

   ```
   Dynamic exception type: class std::runtime_error
   std::exception::what: bad MATLAB Runtime installation:
   C:\Program Files\MATLAB\MATLAB Runtime\v82
   (C:\Program Files\MATLAB\MATLAB Runtime\v82\bin\
   win64\mps_worker_app could not be found)
   ```
3. The message indicates the installation of the MATLAB Runtime is incomplete or has been corrupted. To solve this problem, reinstall the MATLAB Runtime.

Server Diagnostic Tools

In this section...

Log Files

Each server writes a log file containing data from both the main server process, as well as the workers, named `server_name/log/main.log`. You can change the primary log folder name from the default value (`log`) by setting the option `log-root` in `main_config`.

The primary log folder contains the `main.log` file, as well as a symbolic link to this file with the auto-generated name of `main_date_fileID.log`.

The `stdout` stream of the main server process is captured as `log/main.out`.

The `stderr` stream of the main server process is captured as `log/main.err`.

Process Identification Files (PID Files)

Each process that the server runs generates a *Process Identification File (PID File)* in the folder identified as `pid-root` in `main_config`.

The main server PID file is `main.pid`; for each MATLAB Runtime worker process, it is `worker-n.pid`, where `n` is the unique identifier of the worker.

PID files are automatically deleted when a process exits.

Endpoint Files

Endpoint files are generated to capture information about the server’s bound external interfaces. The files are created when you start a server instance and deleted when you stop it.

server_name/endpoint/http contains the IP address and port of the clients connecting to the server. This information can be useful in the event that zero (0) is specified in main_config, indicating that the server bind to a free port.

Manage Log Files

In this section...

Best Practices for Log Management

Use these recommendations as a guide when defining values for the options listed in "Log Retention and Archive Settings" on page 4-8.

- Avoid placing `log-root` and `log-archive-root` on different physical file systems.
- Place log files on local drives, not on network drives.
- Send MATLAB output to `stdout`. Develop an appropriate, consistent logging strategy following best MATLAB coding practices. See *MATLAB Programming Fundamentals* for guidelines.

Log Retention and Archive Settings

Log data is written to the server's `main.log` file for as long as a specific server instance is active, or until midnight. When the server is restarted, log data is written to an archive log, located in the archive log folder specified by `log-archive-root`.

You can set parameters that define when `main.log` is archived using the following options in each server's `main_config` file.

- `log-rotation-size` — When `main.log` reaches this size, the active log is written to an archive log (located in the folder specified by `log-archive-root`).
- `log-archive-max-size` — When the combined size of all files in the archive folder (location defined by `log-archive-root`) reaches this limit, archive logs are purged until the combined size of all files in the archive folder is less than `log-archive-max-size`. Oldest archive logs are deleted first.

Specify values for these options using the following units and notations:

Represent these units of measure...	Using this notation...	Example
Byte	`b`	`900b`
Kilobyte (1024 bytes)	`k`	`700k`
Megabytes (1024 kilobytes)	`m`	`40m`
Gigabytes (1024 megabytes)	`g`	`10g`
Terabytes (1024 gigabytes)	`t`	`2t`
Petabytes (1024 terabytes)	`p`	`1p`

Note: The minimum value you can specify for `log-rotation-size` is 1 megabyte.

On Windows 32-bit systems, values larger than 2^{32} bytes are not supported. For example, specifying `5g` is not valid on Windows 32-bit systems.

Setting Log File Detail Levels

The log level provides different levels of information for troubleshooting:

- `error` — Notification of problems or unexpected results.
- `warning` — Events that could lead to problems if unaddressed.
- `information` — High-level information about major server events.
- `trace` — Detailed information about the internal state of the server.

The log level is set using the `log-severity` configuration property.

Before you call support, you should set logging levels to `trace`.

Common Error Messages and Resolutions

In this section...

(404) Not Found

Commonly caused by requesting a component that is not deployed on the server, or trying to call a function that is not exported by the given component.

Verify that the name of the deployable archive specified in your `Uri` is the same as the name of the deployable archive hosted in your `auto_deploy` folder.

Error: Bad MATLAB Runtime Instance

Common causes of this message include:

- You are not properly qualifying the path to the MATLAB Runtime. You must include the version number. For example, you need to specify:

  `C:\Program Files\MATLAB\MATLAB Runtime\v`*`n.n`*
  not

  `C:\Program Files\MATLAB\MATLAB Runtime`

Error: Server Instance not Specified

MATLAB Production Server can't find the server you are specifying.

Ensure you are either entering commands from the folder containing the server instance, or are using the `-C` command argument to specify a precise location of the server instance.

For example, if you created `server_1` in `C:\tmp\server_1`, you would issue the `mps-start` command from within that folder to avoid specifying a path with the `-C` argument:

```
cd c:\tmp\server_1
mps-start server_1
```

For more information, see "Start a Server Instance" on page 1-12.

Error: invalid target host or port

The port number specified has not been properly defined to your computer. Define a valid port and retry the command.

Error: HTTP error: HTTP/x.x 404 Component not found

This error can be caused by a number of reasons. Consult the "Log Files" on page 4-6 for further details on the precise cause of the problem.

Commands — Alphabetical List

mps-check

Test and diagnose a MATLAB Production Server instance for problems

Syntax

`mps-check` [--timeout *`seconds`*] *`host`*:*`port`*

Description

`mps-check` sends a request to a MATLAB Production Server instance and receives a status report that is used to identify issues that cause the product to run less than optimally.

Information reported by `mps-check` to `stdout` include:

- Status of the server instance
- Port the HTTP interface is listening on
- Deployed archives for a server instance

Before using `mps-check`, you must deploy *`mcrroot`*`/bin/`*`arch`*`/mps_check.ctf` to the server instance.

- *`mcrroot`* is the path to the MATLAB Runtime installation folder.
- *`arch`* is standard abbreviation for the system's operating system and hardware architecture.

Input Arguments

- `--timeout` *`seconds`* — The time, in seconds, to wait for a response from the server before timing out. The default is two minutes.
- *`host`* — The host name of the machine running the server instance.
- *`port`* — The port number on which the server instance listens for requests.

Definitions

Server Instance

An instance of the MATLAB Production Server. The files contained in the folder created by `mps-new`, defined by *`path`*`/`, comprise one configuration of the MATLAB Production Server product.

Examples

Display diagnostic information for the server instance running on port 9910 of the local computer.

```
mps-check localhost:9910
```

```
Connecting to localhost:9910
Connected
Sending HTTP request
HTTP request sent
HTTP response received
MPS status check completed successfully
```

Introduced in R2012b

mps-license-reset

Force a server instance to immediately attempt license checkout

Syntax

```
mps-license-reset [-C path/]server_name
```

Description

`mps-license-reset [-C path/]server_name` triggers the server to checkout a license immediately, regardless of the current license status. License keys that are currently checked out are checked in first.

Input Arguments

`-C path/`

Specify a path to the server instance. If this option is omitted, the current working folder and its parents are searched to find the server instance.

`server_name`

Server checking out license

Examples

Create a new server instance and display the status of each folder in the file hierarchy, as the server instance is created:

```
mps-license-reset -C /tmp/server_2
```

See Also

`mps-status`

Introduced in R2012b

mps-new

Create a server instance

Syntax

```
mps-new [path/]server_name [-v] [--service] [--service-name name]
[--service-description description] [--service-user user] [--
service-password password] [--noprompt]
```

Description

`mps-new [path/]server_name [-v] [--service] [--service-name name] [--service-description description] [--service-user user] [--service-password password] [--noprompt]` makes a new folder at *`path`* and populates it with the default folder hierarchy for a server instance.

Tips

- Before creating a server instance, ensure that no file or folder with the specified *`path`* currently exists on your system.
- After issuing `mps-new`, issue `mps-start` to start the server instance.

Input Arguments

`path`

Path to server instance.

`server_name`

Name of the server instance to create.

If you are creating a server instance in the current working folder, you do not need to specify a full path; specify only the server name.

`-v`

Display the status of each folder in the file hierarchy created to form a server instance

`--service`

On Windows, register the server instance as a Windows service.

The Windows service default settings are:

- Service Display Name: MATLAB Production Server – *`path\server_name`*
- Service Description: MATLAB Production Server running instance *`path\server_name`*
- Service User: LocalSystem

The Windows service is configured to start when the machine starts, not at creation of the service. After you have made configuration changes, start the server instance using `mps-start`.

`--service-name` ***`name`***

Display name for the Windows service associated with the server instance

`--service-description` ***`description`***

Informational statement describing the Windows service associated with the server instance

`--service-user` ***`user`***

Windows account under which the service associated with the server instance should run. The user account must have read, write, and, delete permissions for the instance directory as well read and execute permissions for the MATLAB Production Server installation directory.

`--service-password` ***`password`***

Password for the service user account

`--noprompt`

Indicates that no prompts are generated

Examples

Create a Server Instance

Create a new server instance, and display the status of each folder in the file hierarchy, as the server instance is created:

```
mps-new /tmp/server_1 -v
```

```
server_1/.mps-version...ok
server_1/config/...ok
server_1/config/main_config...ok
server_1/endpoint/...ok
server_1/auto_deploy/...ok
server_1/.mps-socket/...ok
server_1/log/...ok
server_1/pid/...ok
```

Create a Windows Service

Create a new server instance, and register it as a Windows service:

```
mps-new /tmp/server_1 --service
```

See Also

`mps-start` | `mps-status`

More About

- “Server Overview” on page 1-2

Introduced in R2012b

mps-profile

Turn profiling on or off

Syntax

`mps-profile` [-C [*path*/]*instance_name*] {on | off} [*object*...]

Description

`mps-profile` turns profiling on or off for specified objects.

Input Arguments

- `-C` — Specify a path to the server instance. If this option is omitted, the current working folder and its parents are searched to find the server instance.
- `on` — Activate profiling.
- `off` — Deactivate profiling.
- *`object`* — The list of objects whose profiling behavior is changed.

  Valid object values are:

  - `requests`
  - `worker_pool`

  If no object is specified the command changes all objects.

Examples

Turn profiling on.

```
mps-profile on
```

Turn request profiling on without turning on worker pool profiling.

```
mps-profile on requests
```

Introduced in R2015b

mps-restart

Stop and start a server instance

Syntax

```
mps-restart [-C [path/]server_name] [-f]
```

Description

`mps-restart [-C [path/]server_name] [-f]` stops a server instance, then restarts the same server instance. Issuing `mps-restart` is equivalent to issuing the `mps-stop` and `mps-start` commands in succession.

Tips

- After issuing `mps-restart`, issue the `mps-status` command to verify the server instance has started.
- If you are restarting a server instance in the current working folder, you do not need to specify a full path. Only specify the server name.

Input Arguments

`-C path/`

Specify a path to the server instance. If this option is omitted, the current working folder and its parents are searched to find the server instance. If you are restarting a server instance in the current working folder, you do not need to specify a full path. Only specify the server name.

`server_name`

Name of the server to be restarted.

`-f`

Force success even if the server instance is stopped. Restarting a stopped instance returns an error.

Examples

Restart a server instance named `server_1`, located in folder `tmp`. Force successful completion of `mps-restart`.

```
mps-restart -f -C /tmp/server_1
```

See Also

`mps-start` | `mps-status` | `mps-stop`

Introduced in R2012b

mps-service

Create or modify a Windows service for a server instance

Syntax

```
mps-service [-C [path/]server_name] create [--name name] [--
description description] [--user user] [--password password] [--
noprompt]

mps-service [-C [path/]server_name] update [--name name] [--
description description] [--user user] [--password password] [--
instance-root new_path] [--noprompt]

mps-service [-C [path/]server_name] delete
mps-service delete service_name [[--force]|[-f]]
mps-service clean [[--force]|[-f]][[--verbose]|[-v]]

mps-service [-C [path/]server_name] undelete

mps-service [-C [path/]server_name]
mps-service list
```

Description

`mps-service [-C [path/]server_name] create [--name name] [--description description] [--user user] [--password password] [--noprompt]` creates a Windows service for the server instance.

The Windows service default settings are:

- Service Display Name: MATLAB Production Server – *`path\server_name`*
- Service Description: MATLAB Production Server running instance *`path\server_name`*
- Service User: LocalSystem

The Windows service is configured to start when the machine starts, not at creation of the service. After you have made configuration changes, start the server instance using `mps-start`.

`mps-service [-C [path/]server_name] update [--name name] [--description description] [--user user] [--password password] [--instance-root new_path] [--noprompt]` updates the Windows service entry for the server instance.

`mps-service [-C [path/]server_name] delete` deletes the Windows service entry for the server instance.

`mps-service delete service_name [[--force]|[-f]]` deletes the Windows service entry by name.

`mps-service clean [[--force]|[-f]][[--verbose]|[-v]]` deletes invalid Windows service entries.

Invalid Windows service entries are entries where either the target version of MATLAB Production Server is not present or the associated server instance no longer exists.

`mps-service [-C [path/]server_name] undelete` restores the deleted Windows service entry for the server instance.

`mps-service [-C [path/]server_name]` displays the Windows service entry for the server instance.

`mps-service list` lists the Windows service entries for all server instances.

Input Arguments

`-C` *`path`*`/`

Path to server instance

`server_name`

Name of the server instance

`--name` *`name`*

Display name for the Windows service associated with the server instance

`--description` *`description`*

Informational statement describing the Windows service associated with the server instance

`--user` *`user`*

Windows account under which the service associated with the server instance should run. The user account must have read, write, and, delete permissions for the instance directory as well read and execute permissions for the MATLAB Production Server installation directory.

`--password` *`password`*

Password for the service user account

`--instance-root` *`new_path`*

Updated path to server instance

`--noprompt`

Indicate that no prompts are generated

`--force`, `-f`

Force deletion without prompting

`--verbose`, `-v`

Include details about why the service is not valid.

Examples

Create a Windows Service

Create a default Windows service for the server instance `server_1`:

```
mps-service -C tmp/server_1 create
```

Delete a Windows Service

Delete the Windows service entry for the server instance `server_1`:

```
mps-service -C tmp/server_1 delete
```

List Existing Windows Services

List the Windows service entries for all the server instances installed on the local machine:

```
mps-service list
```

```
Service Name:  MATLAB Production Server {01234567-89ab-cdef-0123-456789abcdef}
Display Name:  MATLAB Production Server - My Custom Name
Description:   My Description
Instance Root: C:\instances\instance1
MPS Root:      C:\Program Files\MATLAB\MATLAB Production Server\R2014b
Status:        Started

Service Name:  MATLAB Production Server {01234567-89ab-cdef-0123-456789abcdef}
Display Name:  MATLAB Production Server - c:\instances\instance2
Description:   MATLAB Production Server running instance C:\instances\instance2
Instance Root: C:\instances\instance2
MPS Root:      C:\Program Files\MATLAB\MATLAB Production Server\R2015a
Status:        Stopped
```

See Also

`mps-new`

Introduced in R2015a

mps-setup

Set up a server environment

Syntax

```
mps-setup [mcrroot]
```

Description

`mps-setup [mcrroot]` sets location of MATLAB Runtime and other start-up options.

The `mps-setup` command sets the default path to the MATLAB Runtime for all server instances you create with the product. This is equivalent to presetting the `--mcr-root` option in each server's `main_config` configuration file.

If a default value already exists in *`server_name`*`/config/mcrroot`, it is updated with the value specified when you run the command line wizard.

Tips

- Run `mps-setup` from the script folder. Alternatively, add the `script` folder to your system `PATH` environment variable to run `mps-setup` from any folder on your system.
- Run `mps-setup` without arguments and it will search your system for MATLAB Runtime instances you may want to use with MATLAB Production Server.
- Run `mps-setup` by passing the path to the MATLAB Runtime as an argument. This method is ideal for non-interactive (silent) installations.

Input Arguments

`mcrroot`

Specify a path to the MATLAB Runtime if running `mps-setup` in non-interactive, or silent, mode.

Examples

Run `mps-setup` non-interactively, by passing in a path to the MATLAB Runtime instance that you want MATLAB Production Server to use.

```
mps-setup "C:\Program Files\MATLAB\MATLAB Runtime\mcrver"
```

`mcrver` is the version of the MATLAB Runtime to use.

See Also

`mps-new` | `mps-start` | `mps-status`

Introduced in R2012b

mps-start

Start a server instance

Syntax

```
mps-start [-C [path/]server_name] [-f]
```

Description

`mps-start [-C [path/]server_name] [-f]` starts a server instance

Tips

- After issuing `mps-start`, issue the `mps-status` command to verify the server instance has `STARTED`.
- If you are starting a server instance in the current working folder, you do not need to specify a full path. Only specify the server name.

Input Arguments

`-C path/`

Specify a path to the server instance. If this option is omitted, the current working folder and its parents are searched to find the server instance.

`server_name`

Name of the server to be started.

`-f`

Force success even if the server instance is currently running. Starting a running server instance is considered an error.

Examples

Start a server instance named `server_1`, located in folder `tmp`. Force successful completion of `mps-start`.

```
mps-start -f -C /tmp/server_1
```

See Also

`mps-new` | `mps-restart` | `mps-status` | `mps-stop`

More About

- “Start a Server Instance” on page 1-12
- “Server Overview” on page 1-2

Introduced in R2012b

mps-status

Display status of a server instance

Syntax

```
mps-status [-C [path/]server_name][--statistics|-s
[sample_interval]] [--json|-j]
```

Description

`mps-status [-C [path/]server_name][--statistics|-s [sample_interval]] [--json|-j]` displays the status of the server (`STARTED`, `STOPPED`), along with a full path to the server instance. Additionally, it can display performance statistics about the server including:

- sample interval in milliseconds
- CPU utilization
- number of active worker processes
- number of requests in queue
- memory usage
- request throughput per second
- total queue time in milliseconds

Input Arguments

`-C path/`

Specify a path to the server instance. If this option is omitted, the current working folder and its parents are searched to find the server instance.

`server_name`

Server to be queried for status

`--statistics [`*`sample_interval`*`], -s [`*`sample_interval`*`]`

Specify that statistics are to be collected and displayed.

The optional *`sample_interval`* allows you to specify the interval, in milliseconds, over which statistics are collected. The default is 500.

Note: If you specify a sample interval of 0, only one sample is taken. Two samples are required to compute some statistics such as CPU utilization and throughput.

`--json, -j`

Specify that statistics are output in JSON format:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Instance Status",
    "description": "Status and Statistics for a MATLAB Production
                    Server Instance",
    "type": "object",
    "properties": {
        "instancePath": {
            "description": "Filesystem path for the server
                            instance",
            "type": "string"
        },
        "started": {
            "type": "boolean"
        },
        "license": {
            "type": "object",
            "properties": {
                "status": {
                    "enum": [ "CHECKED_OUT", "IN_GRACE_PERIOD",
                              "GRACE_PERIOD_EXPIRED" ]
                },
                "type": {
                    "enum": [ "INVALID", "UNKNOWN", "COMPILED",
                              "TRIAL", "EDU", "COMM" ]
                },
                "number": {"type": "string"}
            },
```

```
    "required": ["status"]
},
"statistics": {
    "type": "object",
    "properties": {
        "sampleIntervalMS": {
            "description": "The difference in upTime
                            between the two samples, 0 if
                            only a single sample was
                            taken",
            "type": "number"
        },
        "localTime": {
            "description": "Local Time at server in format
                            YYYY.MM.DD HH.MM.SS.SSSSSS",
            "type": "string"
        },
        "upTime": {
            "description": "Time since server start in
                            fractional seconds",
            "type": "number"
        },
        "cpuTime": {
            "description": "CPU time consumed by all server
                            processes in fractional
                            seconds",
            "type": "number"
        },
        "cpuPercentage": {
            "description": "CPU utilzation, computed using
                            change in cpuTime and upTime
                            between two samples",
            "type": "number"
        },
        "totalRequestsReceived": {
            "description": "The number of valid requests
                            received",
            "type": "integer"
        },
        "totalRequestsStarted": {"type": "integer"},
        "totalRequestsFailedToStart": {
            "description": "The number of requests that
                            could not be started",
            "type": "integer"
```

```
                },
                "totalRequestsFinishedHttpSuccess": {
                    "type": "integer"
                },
                "totalRequestsFinishedHttpError": {
                    "description": "Note: does not includes
                                    requests that failed to start",
                    "type": "integer"
                },
                "memoryWorkingSet": {
                    "description": "Amount of memory resident in
                                    physical memory for all
                                    processes (KiB)",
                    "type": "number"
                },
                "throughput": {
                    "description": "Requests retired per second,
                                    computed using the number of
                                    requests finished or failed to
                                    start over two samples",
                    "type": "number"
                },
                "totalQueueTimeMS": {
                    "description": "Sum of the wait times for
                                    currently queued requests",
                    "type": "number"
                }
            }
        }
    },
    "required": ["instancePath", "started"]
}
```

Examples

Check if a Server is Running

Display status of server instance `server_1`, residing in `tmp` folder.

```
mps-status -C /tmp/server_1
```

If server is running and running with a valid license:

```
'/tmp/server_1' STARTED
license checked out
```

If server is not running:

```
'/tmp/server_1' STOPPED
```

Report Statistics in a Human Readable Format

Display statistics for the server instance `server_1`, residing in `tmp` folder.

```
mps-status -C /tmp/server_1 -s
```

If server is running and running with a valid license:

```
'/tmp/server_1' STARTED
license checked out
Statistics:
Sample Interval (ms):     500
CPU Utilization (%):      40
Active Worker Processes:  2
Requests in Queue:        1
Memory Usage (KiB):       1024
Throughput (requests/s):  10
Total Queue Time (ms):    100
```

Report Statistics in JSON Format

Display statistics for the server instance `server_1`, residing in `tmp` folder.

```
mps-status -C /tmp/server_1 -s -j
```

If server is running and running with a valid license:

```
{
    "instancePath":"L:\\MPS\\stats",
    "license":{
        "number":"unknown",
        "status":"CHECKED_OUT",
        "type":"COMM"
    },
    "started":true,
    "statistics":{
        "cpuPercentage":0,
        "cpuTime":1.7628113000000001,
```

```
        "localTime":"2015.04.28 16:52:49.874483",
        "memoryWorkingSet":393468,
        "sampleIntervalMS":500.31748899999951,
        "throughput":0,
        "totalQueueTimeMS":0,
        "totalRequestsFailedToStart":0,
        "totalRequestsFinishedHttpError":0,
        "totalRequestsFinishedHttpSuccess":0,
        "totalRequestsReceived":0,
        "totalRequestsStarted":0,
        "upTime":6.9780032949999997
    }
}
```

See Also

`mps-restart` | `mps-start` | `mps-stop` | `mps-which`

More About

- “Start a Server Instance” on page 1-12
- “Server Overview” on page 1-2
- “License Server Status Information” on page 4-2

Introduced in R2012b

mps-stop

Stop a server instance

Syntax

```
mps-stop [-C [path/]server_name] [-f] [-v] [--timeout hh:mm:ss]
```

Description

`mps-stop [-C [path/]server_name] [-f] [-v] [--timeout hh:mm:ss]` closes HTTP server socket and all open client connections immediately. All function requests that were executing when the command was issued are allowed to complete before the server shuts down.

Tips

- After issuing `mps-stop`, issue the `mps-status` command to verify the server instance has `STOPPED`.
- If you are stopping a server instance in the current working folder, you do not need to specify a full path. Only specify the server name.
- Note that the timeout option (`--timeout` *`hh`*:*`mm`*:*`ss`*) is specified with two (2) dashes, not one dash.

Input Arguments

`-C` *`path`*`/`

Specify a path to the server instance. If this option is omitted, the current working folder and its parents are searched to find the server instance.

`server_name`

Name of the server to be stopped.

`-f`

Force success even if the server instance is not currently stopped. Stopping a stopped instance is considered an error.

`-v`

Displays system messages relating to termination of server instance.

`--timeout hh:mm:ss`

Set a limit on how long `mps-stop` will run before returning either success or failure. For example, specifying `--timeout 00:02:00` indicates that `mps-stop` should exit with an error status if the server takes longer than two (2) minutes to shut down. The instance continues to attempt to terminate even if `mps-stop` times out. If this option is not specified, the default behavior is to wait as long as necessary (infinity) for the instance to stop.

Examples

Stop server instance `server_1`, located in `tmp` folder. Force successful completion of `mps-stop`. Timeout with an error status if `mps-stop` takes longer than three (3) minutes to complete.

In this example, the verbose (`-v`) option is specified, which produces an output status message.

```
mps-stop -f -v -C /tmp/server_1 --timeout 00:03:00
```

Example Output

```
waiting for stop... (timeout = 00:03:00)
```

See Also

`mps-new` | `mps-restart` | `mps-start` | `mps-status`

Introduced in R2012b

mps-support-info

Display licensing and configuration information of a MATLAB Production Server instance

Syntax

`mps-support-info` [-C [*path*/]*server_name*]

Description

`mps-support-info` displays licensing and configuration information of a MATLAB Production Server instance.

Input Arguments

- *path* — The path to where the server instance is installed.
- *server_name* — The name of the server instance to locate in the current folder.

Examples

Display licensing and configuration information of server instance `fred`, residing in `/` folder.

```
mps-support-info -C /fred
```

```
Instance Version:           1.0
License Number:             UNKNOWN -- MPS stopped
MPS Version:                UNKNOWN -- MPS stopped
Available License Number:   857812
Client Version:             1.0.1 R2013a
Operating System:           Microsoft Windows 7 Enterprise Edition (build 7601), 64-bit
Number of CPU cores:        8
CPU Info:                   Intel(R) Xeon(R) CPU           W3550  @ 3.07GHz 64-bit Compatible
Memory:                     11.9915 GB ( 1.2574e+007 KB )
```

Introduced in R2012b

mps-which

Display path to server instance that is currently using the configured port

Syntax

```
mps-which [-C [path/]server_name]
```

Description

`mps-which [-C [path/]server_name]` is useful when running multiple server instances on the same machine. If you attempt to start two server instance on the same port, the latter server instance will fail to start, displaying an `address-in-use` error. `mps-which` identifies which server instance is using the port.

Tips

- If you are creating a server instance in the current working folder, you do not need to specify a full path. Only specify the server name.

Input Arguments

`-C path/`

Specify a path to the server instance. If this option is omitted, the current working folder and its parents are searched to find the server instance.

`server_name`

Server to be queried for path.

Examples

`server_1` and `server_2`, both residing in folder `tmp`, are configured to use to same port, defined by the `http` configuration property.

Run `mps-which` for both servers:

```
mps-which -C /tmp/server_1
```

```
mps-which -C /tmp/server_2
```

Example Output

In both cases, the server that has allocated the configured port displays (`server_1`):

```
/tmp/server_1
```

See Also

`mps-status`

Introduced in R2012b

6

Configuration Properties—Alphabetical List

auto-deploy-root

Folder the server instance scans for deployable archives

Syntax

```
--auto-deploy-root path
```

Description

`--auto-deploy-root` *`path`* specifies the folder the server instance scans for deployable archives. Deployable archives placed in this folder are automatically unpacked and deployed when the instance is started. No restart is necessary when a deployable archive is added, updated, or removed. Many instances may share a single `auto-deploy-root`. Using this folder allows near-simultaneous hot deployment to multiple instances. The folder is scanned every five seconds for changes.

Parameters

`path`

Path to the folder scanned for deployable archives relative to the server instance's root folder.

Examples

Scan the `auto_deploy` folder for deployable archives to hot deploy.

```
--auto-deploy-root ./auto_deploy
```

cors-allowed-origins

Specify the domain origins from which clients are allowed to make requests to the server

Syntax

```
--cors-allowed-origins [ LIST | * ]
```

Description

`cors-allowed-origins` specifies the set of domains origins from which clients are allowed to make requests to a MATLAB Production Server instance. Cross-Origin Resource Sharing or CORS defines a way in which client-side web applications and a server can interact to safely determine whether or not to allow a cross-origin request. Most clients such as browsers use the `XMLHttpRequest` object to make a cross-domain request. This is especially true for client code written using JavaScript®. For MATLAB Production Server to support such requests `cors-allowed-origins` must be enabled on the server.

Parameters

`*`

Requests from any domain origin are allowed access to the sever.

`LIST`

Requests from a list of comma-separated domain origins are allowed access to the server.

Examples

Requests from any domain origin are allowed access to the sever.

```
--cors-allowed-origins *
```

Requests from a specific list of domain origins are allowed access to the server.

```
--cors-allowed-origins http://www.w3.org, https://www.apache.org
```

See Also

`http`

disable-control-c

Disable keyboard interruptions for server instance

Syntax

```
--disbable-control-c
```

Description

`disable-control-c` disables keyboard interruption for the server instance. The server instance does not respond to **CTRL-C**.

Examples

Disable the **CTRL-C** button.

```
--disable-control-c
```

endpoint-root

Folder used to store server named endpoints

Syntax

```
--endpoint-root path
```

Description

`--endpoint-root` *`path`* specifies the location for storing server named endpoints. Each interface used to communicate with the outside world generates an endpoint file in this folder. Normally that means:

- http - The HTTP function execution interface.
- control - The local control interface used by the scripting commands.

These files contain the `host:post` portion of the URL used to communicate with the named service.

Note: While modifying this location is allowed, each instance must have a unique endpoint directory; otherwise behavior is undefined.

Parameters

`path`

Path to the folder used to store endpoint files relative to the server instance's root folder.

Examples

Store endpoint files in the `endpt` folder.

```
--endpoint-root ./endpt
```

extract-root

Root folder used to store contents of deployed archives

Syntax

```
--extract-root path
```

Description

`--extract-root` *`path`* specifies the root folder used to store the expanded contents of the deployable archives deployed on the server instance. Deployable archives are unpacked to a hidden subdirectory of `extract-root`.

Parameters

`path`

Path to the root folder used to store contents of deployable archives relative to the server instance's root folder.

Examples

Extract deployable archives into the `archives` folder.

```
--extract-root ./archives
```

hide-matlab-error-stack

Hide the MATLAB stack from the clients

Syntax

```
--hide-matlab-error-stack
```

Description

`hide-matlab-error-stack` controls whether the MATLAB stack is exposed to the client. The stack can be sent to the client during development and debug phase, but can be turned off in production.

Examples

Do not transmit the error stack to clients.

```
--hide-matlab-error-stack
```

http

URL that server instance uses for insecure connections

Syntax

```
--http host:port
```

Description

`http` specifies the interface port and optional address or host name.

Parameters

`host`

Host name, or IP address, of the machine running the server instance. If not specified, the server binds to any available interface.

`port`

Port number the server instance uses to accept connections. `0` specifies bind to any available port.

Examples

Restrict access to the HTTP interface to local clients only, port 9910.

```
--http localhost:9910
```

Bind to any free port. The bound address is written to `$INSTANCE/endpoint/https`.

```
--http 0
```

Bind to a specific IP address and port.

```
--http 234.27.101.3:9920
```

Bind to a specific host name on any free port

```
--http my.hostname.com:0
```

http-linger-threshold

Amount of data the server instance discards after an HTTP error and before the server instance closes the TCP connection

Syntax

```
--http-linger-threshold size
```

Description

`http-linger-threshold` sets the amount of data a server instance reads after an error. If an HTTP request is rejected and the server instance sends back an HTTP error response such as HTTP 404/413, the server instance does not close the TCP connection immediately. Instead it waits for the client to shut down the TCP connection. This ensures that the client receives the HTTP error response sent by the server instance. During this time, the server instance receives, and discards, data from the client, until the amount of data received equals `http-linger-threshold`. After that, the server instance resets the TCP connection.

By default, the threshold is unlimited and the server instance waits to receive the whole HTTP request.

Parameters

`size`

Amount of data received before the TCP connection is reset.

Examples

Set the linger threshold to be 64 MB.

```
--http-linger-threshold 64MB
```

Set the linger threshold to be 32 KB.

```
--http-linger-threshold 32KB
```

Set the linger threshold to be 1024 B.

```
--http-linger-threshold 1024
```

https

URL that server instance uses for secure connections

Syntax

```
--https host:port
```

Description

`https` specifies the interface port and optional address or host name.

Parameters

`host`

Host name, or IP address, of the machine running the server instance. If not specified, the server binds to any available interface.

`port`

Port number the server instance uses to accept connections. `0` specifies bind to any available port.

Examples

Restrict access to the HTTP interface to local clients only, port 9920.

```
--https localhost:9920
```

Bind to any free port. The bound address is written to `$INSTANCE/endpoint/https`.

```
--https 0
```

Bind to a specific IP address and port.

```
--https 234.27.101.3:9920
```

Bind to a specific host name on any free port

```
--https my.hostname.com:0
```

license

Locations searched for valid licenses

Syntax

```
--license pathList
```

Description

`license` specifies the license servers or the license files used by the server instance. You can specify multiple license sources with this option.

If this option is not specified, the server searches in the default locations for the license files.

Parameters

`pathList`

Path to one or more license servers or license files. Multiple entries are separated by the appropriate path separator for the platform.

Examples

A Unix server looks for licenses using a license server hosted on port 27000 of `hostA` and in `/opt/license/license.dat`.

```
--license 27000@hostA
--license /opt/license/license.dat
```

The same configuration in one line.

```
--license 27000@hostA:/opt/license/license.dat
```

A Windows server looks for licenses using a license server hosted on port 27000 of `hostA` and in `c:\license\license.dat`.

```
--license 27000@hostA
--license c:\license\license.dat
```

The same configuration in one line.

```
--license 27000@hostA;c:\license\license.dat
```

license-grace-period

Maximum length of time the server instance responds to HTTP requests after license server heartbeat has been lost

Syntax

```
--license-grace-period hr:min:sec.fractSec
```

Description

`license-grace-period` specifies the grace period, which starts at the first heartbeat loss event. Once the grace period expires, the server instance rejects any new incoming HTTP requests.

The default grace period is 2 hours 30 minutes. The maximum value is 2 hours 30 minutes. The minimum value is 10 minutes.

Parameters

hr

Hours in interval.

min

Minutes in interval.

sec

Seconds in interval.

fractSec

Fractional seconds in interval.

Examples

The grace period lasts for 1 hour, 29 minutes, 5 seconds.

```
--license-grace-period 1:29:05
```

The grace period lasts for 10 minutes and 250 ms.

```
--license-grace-period 00:10:00.25
```

license-poll-interval

Interval of time before license server is polled to verify and check out a valid license after the grace period expires

Syntax

```
--license-poll-interval hr:min:sec.fractSec
```

Description

`license-poll-interval` specifies interval at which the server instance polls the license server after the license server has timed out or after the grace period has expired.

The default poll interval is 10 minutes. The minimum value is 10 minutes.

Parameters

`hr`

Hours in interval.

`min`

Minutes in interval.

`sec`

Seconds in interval.

`fractSec`

Fractional seconds in interval.

Examples

Poll for licenses at intervals of 1 hour, 29 minutes, 5 seconds.

```
--license-poll-interval 1:29:05
```

Poll for licenses at intervals of 10 minutes and 250 ms.

```
--license-poll-interval 00:10:00.25
```

log-archive-max-size

Maximum size of the log archive folder

Syntax

```
--log-archive-max-size size
```

Description

`log-archive-max-size` specifies the maximum size to which the log archive folder can grow before old log files are deleted.

If this property is not specified, then the log archive grows without limit.

Parameters

size

Size, in bytes, of the archive folder.

Examples

Reap log archives when they reach 5 MB.

```
--log-archive-max-size 5MB
```

log-archive-root

Path to the folder containing archived log files

Syntax

`--log-archive-root` *`path`*

Description

`--log-archive-root` *`path`* specifies the path to directory that stores rotated log files.

Note: If you omit this property, rotated logs remain in the log root directory, which grows unbounded as logs are rotated.

Parameters

`path`

Path to the folder where log files are archived relative to the server instance's root folder.

Examples

Archive logs to *`server_root`*`/old_logs`.

```
--log-archive-root ./old_logs
```

log-handler

Add custom log handler

Syntax

`--log-handler` *`format command`*

Description

`--log-handler` *`format command`* adds a log handler that writes log data to the application specified by *`command`* in the format specified by *`format`*.

The server instance launches an instance of the log handler at startup. All log events are sent to the `STDIN` stream of the log handler. The `STDOUT` and `STDERR` streams of the log handler are captured and written to *`INSTANCE_ROOT`*`/log/custom_logger_`*`N`*`.out` and *`INSTANCE_ROOT`*`/log/custom_logger_`*`N`*`.err`.

Parameters

`format`

Format used to write log events. Valid values are:

- `text/plain`
- `text/json`
- `text/xml`

`command`

Application launched to process log events.

Examples

Send log events to a custom JSON parser that prepares performance graphs.

```
--log-handler text/json perf_grapher
```

log-root

Path to the log file folder

Syntax

`--log-root` *`path`*

Description

`--log-root` *`path`* specifies the location for log files.

When a server instance starts, the following log files are created:

- `main__`*`DATE`*`__`*`SERIAL`*`.log` — The head process main log
- `main.log` — A link to the mostly recently written main log file
- `main.out` — Captured standard output from the main process
- `main.err` — Captured standard error output from the main process

When the server instance stops, the head process main log is moved to the log archive folder.

Note: Omitting this property disables all logging except for `stdout` and `stderr` capture of main.

Parameters

`path`

Path to the folder where log files are stored relative to the root folder of the server instance.

Examples

Archive logs to *`server_root`*`/logs`.

```
--log-root ./logs
```

log-rotation-size

Size at which the log is archived

Syntax

```
--log-rotation-size size
```

Description

`log-rotation-size` specifies the maximum size to which the log can grow before it is rotated into the archive area. If specified as less than 1 MB, a warning is issued and the effective size is increased to 1 MB.

No entry signifies that logs are never archived.

Parameters

`size`

Size, in bytes, of the log file.

Examples

Rotate logs when they reach 5 MB.

```
--log-rotation-size 5MB
```

log-severity

Severity at which messages are logged

Syntax

```
--log-severity level
```

Description

`log-severity` specifies the level of detail at which to add information to the main log.

Parameters

`level`

Severity threshold at which messages are logged. Valid values are:

- `error` — Notification of problems or unexpected results.
- `warning` — Events that could lead to problems if not addressed.
- `information` — High-level information about major server events.
- `trace` — Detailed information about the internal state of the server.

The levels are cumulative; specifying `information` implies `warning` and `error`.

Examples

Enable all log messages.

```
--log-severity trace
```

mcr-root

Location of a MATLAB Runtime installation

Syntax

```
--mcr-root path
```

Description

`mcr-root` specifies the location of an installed MATLAB Runtime instance. If multiple MATLAB Runtime installations are available, then specify each installation on a separate line.

Note: Specifying multiple MATLAB Runtime installations allows one MATLAB Production Server instance to support multiple versions of the MATLAB Runtime. Specifying multiple MATLAB Runtime installations of the same version has no effect on performance.

If multiple `mcr-root` settings are present, then the server uses dynamic worker pool management, where worker processes are started in response to demand and shut down in response to system resource utilization.

The server instance scans the list of provided MATLAB Runtime installations in order from first to last and chooses the first MATLAB Runtime installation capable of processing the request. A MATLAB Runtime installation can process a request if it is compatible with the deployable archive containing the function being evaluated.

Note:

- A server instance should only be configured to use MATLAB Runtime roots on a local file system. Otherwise, a network partition may cause worker processes to fail.
- All values for `mcr-root` must be for the same OS/hardware combination.

Parameters

path

Path to the root folder of the MATLAB Runtime installation.

Note: The special value `mCRuNsETtOKEN` indicates to the `mps-start` command that there is no MATLAB Runtime installation configured for this instance. Running the `mps-start` command results in an error.

Examples

Use the v80 version of the MATLAB Runtime.

```
--mcr-root /usr/local/MCR/v80
```

Use the v80 and v81 versions of the MATLAB Runtime.

```
--mcr-root /usr/local/MCR/v80
--mcr-root /usr/local/MCR/v81
```

Related Examples

- “Specify the MATLAB Runtime for a Server Instance” on page 1-11
- “Support Multiple MATLAB Versions” on page 1-14

num-threads

Number of request-processing threads within the server instance

Syntax

```
--num-threads count
```

Description

`num-threads` sets the size of the thread pool available to process requests. Server instances do not allocate a unique thread to each client connection. Rather, when data is available on a connection, the required processing is scheduled on the pool of threads in the server main process.

The threads in this pool do not directly evaluate MATLAB functions. There is a single thread within each worker process that executes MATLAB code on behalf of the client.

Set this parameter to 1, and increase it only if the expected load consists of a high volume of short-running requests. This strategy ensures that the available processor resources are balanced between MATLAB function evaluation and processing client-server requests. There is usually no benefit to increasing this parameter to more than the number of available cores.

Parameters

`count`

Number of threads available in the thread pool.

This value must be one or greater.

Examples

Create a pool of 10 threads for processing requests.

```
--num-threads 10
```

num-workers

Maximum number of workers allowed to process work simultaneously

Syntax

```
--num-workers count
```

Description

`num-workers` defines the number of concurrent MATLAB execution requests that can be processed simultaneously. It should correspond to the number of hardware threads available on the local host.

If you specify a single value for the `mcr-root` property, this setting determines the fixed size of the worker pool.

If you specify more than one value for the `mcr-root`, this setting specifies a maximum limit on the size of each subpool specific to MATLAB Runtime. There can be more than specified number of worker processes at a time, but at a maximum only the specified number of workers are allowed to be processing a request.

Parameters

`count`

Number of workers available evaluate functions.

This value must be one or greater.

The maximum value is determined by the number of license keys available for MATLAB Production Server.

Examples

Allow 10 workers to process requests at a time.

```
--num-workers 10
```

pid-root

Folder used to store PID files

Syntax

```
--pid-root path
```

Description

`--pid-root` *`path`* specifies the folder used to store PID files. PID files record the system-specific process identifiers for all processes associated with the server instance. This includes:

- `main.pid` — The process identifiers of the server's head process.
- `worker_`*`N`*`.pid` — The process identifiers of each worker process *`N`*.

In some circumstances, `worker_2.pid` may be present when `worker_1.pid` is not. This is a strong indication that `worker_1` crashed and was restarted automatically. You can confirm this by checking the main log file.

The format of these files is a single decimal integer, the process identifier.

Parameters

`path`

Path to the folder used to store PID files relative to the server instance's root folder.

Examples

Store PID files in the `pid` folder.

```
--pid-root ./pid
```

profile

Turn profiling on or off

Syntax

```
--profile state object
```

Description

`profile` turns profiling on or off for different objects.

Note: Activating profiling has a negative impact on performance.

In some circumstances, `worker_2.pid` may be present when `worker_1.pid` is not. This is a strong indication that `worker_1` stopped and was restarted automatically. You can confirm this by checking the main log file.

When profiling is activated, messages similar to the following are included in the log.

```
12 [2014.02.27 10:13:28.075126] [profile] [SERVICE:http-connection]
[endpoint:[::]:9910] [client:163.72.158.2:57611] [request-id:0:1:5]
[type:arrive] [component:mymagic] [function:magic]
Request arrived and was placed in the queue
13 [2014.02.27 10:13:28.087752] [profile] [SERVICE:http-connection]
[endpoint:[::]:9910] [client:163.72.158.2:57611] [request-id:0:1:5]
[type:start] [worker:3] Request started executing on worker-3
...
15 [2014.02.27 10:13:31.397266] [profile] [SERVICE:http-connection]
[endpoint:[::]:9910] [client:163.72.158.2:57611] [request-id:0:1:5]
[type:finish] [status:200] Request completed with HTTP status 200
```

Parameters

`state`

Specifies if profiling is active. Valid values are:

- `on` — Activate profiling.
- `off` — Activate profiling.

`object`

The list of objects to change. Supported objects are:

- `requests`
- `worker_pool`

If no object is specified, all objects are changed.

Examples

Turn on request profiling.

```
--profile on requests
```

Turn on profiling for all objects.

```
--profile on
```

ssl-allowed-clients

MATLAB programs a client can access

Syntax

```
--ssl-allowed-clients client1,...,clientN:archive1,...,archiveN
```

Description

`ssl-allowed-clients` authorizes clients based on the client certificate common name. Only authorized clients can request the evaluation of MATLAB functions.

If there are no archive names following the common name, the client can access all of the deployed archives. Otherwise, the client can access only the specified archives.

Parameters

`client`

Common name of the client.

`archive`

Name of an archive the clients can access.

Examples

Allow `client1` and `client2` to access `magic.ctf` and `helloworld.ctf`. Allow `client3` access to all deployed archives.

```
--ssl-allowed-client client1,client2:magic,helloworld
--ssl-allowed-client client3
```

ssl-ciphers

List of cipher suites to use

Syntax

```
--ssl-ciphers ciphers
```

Description

`ssl-ciphers` provides a list of cipher suites the server instance can use.

Parameters

`ciphers`

Cipher suites the server instance uses for encryption. Valid values are:

- `ALL` — Use all available cipher suites except eNULL.
- `HIGH` — Use all available high encryption cipher suites.
- *`list`* — Comma-separated list of cipher suites to use.

All OpenSSL configuration strings can be passed with the ciphers. This provides finer control over the selected cipher.

Examples

Use only high encryption cipher suites.

```
--ssl-ciphers HIGH
```

Disable the use of ADH ciphers.

```
--ssl-ciphers ALL:!ADH
```

Use the strongest available ECDHE ciphers.

```
--ssl-ciphers ALL:@STRENGTH
```

Disable the use of ADH ciphers and use the strongest available ECDHE ciphers.

```
--ssl-ciphers ALL:!ADH@STRENGTH
```

ssl-protocols

List of allowed SSL protocols

Syntax

```
--ssl-protocols protocols
```

Description

`ssl-protocols` lists the allowed protocols. The default behavior is to attempt to use TLSv1.2. If this property is not set, the server allows all protocols.

Parameters

`protocols`

Comma-separated list of allowed protocols. Valid entries are:

- `TLSv1`
- `SSLv2`
- `SSLv3`

Examples

Allow only TLSv1.

```
--ssl-protocols TLSv1
```

ssl-tmp-ec-param

Elliptical curve used in ECDHE ciphers

Syntax

```
--ssl-tmp-ec-param elliptic_curve_name
```

Description

`--ssl-tmp-ec-param` *`elliptic_curve_name`* specifies the name of the elliptical curve used in ECDHE ciphers.

If this property is not specified, all ECDHE ciphers will be removed from the list of ciphers available for secure communication.

Parameters

`elliptic_curve_name`

Named of curve. All curves supported by OpenSSL are supported.

Examples

Use the prime256v1 curve.

```
--ssl-tmp-ec-param prime256v1
```

ssl-tmp-dh-param

File containing a pregenerated ephemeral DH key

Syntax

```
--ssl-tmp-dh-param path
```

Description

`ssl-tmp-dh-param` specifies the path to the pre-generated ephemeral DH key. If this parameter is not provided, the server instance automatically generates the DH key at start-up. Providing a pre-generated DH key can decrease instance start time.

Parameters

`path`

Path to the pre-generated DH key. Relative and absolute paths are valid.

Examples

The instance loads the DH key from `dh_param.pem` which is located at *`instance_root`*`/x509`.

```
--ssl-tmp-dh-param ./x509/dh_param.pem
```

ssl-verify-peer-mode

Level of client verification the server instance requires

Syntax

```
--ssl-verify-peer-mode mode
```

Description

`ssl-verify-peer-mode` specifies if the server requires clients to present a valid certificate to connect. Server instances can allow clients to connect with or without providing a valid certificate. All requests will still require authorization.

Parameters

`mode`

Mode used to authenticate clients. Valid values are:

- `no-verify-peer` — No peer certificate verification. The client side does not need to provide a certificate.
- `verify-peer-require-peer-cert` — The client must provide a certificate and the certificate will be verified.

The default is `no-verify-peer`.

Examples

Require clients to provide a certificate.

```
--ssl-verify-peer-mode verify-peer-require-peer-cert
```

use-single-comp-thread

Start MATLAB Runtime with a single computational thread

Syntax

```
--use-single-comp-thread
```

Description

`--use-single-comp-thread` specifies that workers start the MATLAB Runtime with a single computational thread.

Examples

Start the MATLAB Runtime with a single computational thread.

```
--use-single-comp-thread
```

worker-memory-check-interval

Interval at which workers are polled for memory usage

Syntax

```
--worker-memory-check-interval hr:min:sec.fractSec
```

Description

`worker-memory-check-interval` specifies how often to poll the memory usage of a worker process. This setting affects the behavior of all other settings that act based on worker memory usage such as `worker-memory-trigger`, `worker-memory-target`, and `worker-restart-memory-limit`.

Parameters

hr

Hours in interval.

min

Minutes in interval.

sec

Seconds in interval.

fractSec

Fractional seconds in interval.

Examples

Check memory usage every one and a half minutes.

```
--worker-memory-check-interval 0:01:30
```

See Also

`worker-restart-memory-limit` | `worker-restart-memory-limit-interval`

Related Examples

- “Control Worker Restarts” on page 1-16

worker-restart-interval

Time interval at which a server instance stops and restarts its workers

Syntax

```
--worker-restart-interval hr:min:sec.fractSec
```

Description

`worker-restart-interval` specifies the interval at which the server instance stops and restarts its worker processes. If this setting is not given, the workers are not restarted in response to time.

Parameters

hr

Hours in interval.

min

Minutes in interval.

sec

Seconds in interval.

fractSec

Fractional seconds in interval.

Examples

Restart workers at intervals of 1 hour, 29 minutes, 5 seconds.

```
--worker-restart-interval 1:29:05
```

Restart workers at intervals of 10 minutes and 250 ms.

```
--worker-restart-interval 00:10:00.25
```

Related Examples

- “Control Worker Restarts” on page 1-16

worker-restart-memory-limit

Size threshold at which to consider restarting a worker

Syntax

```
--worker-restart-memory-limit size
```

Description

`worker-restart-memory-limit` sets the memory usage limit of a worker process. If a worker's working set size exceeds `worker-restart-memory-limit` for an interval of time greater than `worker-restart-memory-limit-interval`, then that worker is restarted.

Parameters

`size`

Amount of memory used by worker.

Examples

Restart any worker whose working set size exceeds 1 GB for more than 1 hour.

```
--worker-restart-memory-limit 1GB
--worker-restart-memory-limit-interval 1:00:00
```

See Also

`worker-memory-check-interval` | `worker-restart-memory-limit-interval`

Related Examples

- “Control Worker Restarts” on page 1-16

worker-restart-memory-limit-interval

Interval for which a worker can exceed its memory limit before restart

Syntax

```
--worker-restart-memory-limit-interval hr:min:sec.fractSec
```

Description

`worker-restart-memory-limit-interval` sets the interval for which a worker process can exceed its memory limit before restart. If a worker's working set size exceeds `worker-restart-memory-limit` for an interval of time greater than `worker-restart-memory-limit-interval`, then that worker is restarted.

Parameters

`hr`

Hours in interval.

`min`

Minutes in interval.

`sec`

Seconds in interval.

`fractSec`

Fractional seconds in interval.

Examples

Restart any worker whose working set size exceeds 1 GB for more than 1 hour.

```
--worker-restart-memory-limit 1GB
--worker-restart-memory-limit-interval 1:00:00
```

See Also

`worker-memory-check-interval` | `worker-restart-memory-limit`

Related Examples

- “Control Worker Restarts” on page 1-16

x509-ca-file-store

File containing the server certificate authority file

Syntax

```
--x509-ca-file-store path
```

Description

`x509-ca-file-store` specifies the CA file to verify peer certificates. This file contains trusted certificates and certificate revocation lists.

You can also put intermediate certificates into the CA file. An intermediate certificate in the CA file becomes a trusted certificate.

Parameters

`path`

Path to the certificate CA file store. Relative and absolute paths are valid.

Examples

The instance loads the CA store from `ca_file.pem` which is located at *`instance_root`*`/x509`.

```
--x509-ca-file-store ./x509/ca_file.pem
```

x509-cert-chain

File containing the server certificate chain

Syntax

```
--x509-cert-chain path
```

Description

`x509-cert-chain` specifies the file storing the server certificate chain file. It contains one or more PEM formatted certificates. The chain begins with the server's certificate. The server's certificate is followed by the chain of untrusted certificates. To use the certificate chain file, specify `x509-private-key`.

Note: Trusted certificates should not be put into this file.

Parameters

path

Path to the certificate chain file. Relative and absolute paths are valid.

Examples

The instance loads the CA store from `cert_chain.pem` which is located at *instance_root*`/x509`.

```
--x509-cert-chain ./x509/cert_chain.pem
```

x509-passphrase

File containing the passphrase that decodes the private key

Syntax

```
--x509-passphrase path
```

Description

`x509-passphrase` specifies the path to the file containing the passphrase of the encrypted private-key. This is required if `x509-private-key` is specified and the private key file is encrypted. Otherwise, the private key fails to load.

Note: This file must be owner read-only.

Parameters

path

Path to the passphrase file. Relative and absolute paths are valid.

Examples

The instance loads the passphrase from `key_passphrase.pem` which is located at *instance_root*`/x509`.

```
--x509-passphrase ./x509/key_passphrase.pem
```

x509-private-key

File containing the PEM formatted private key

Syntax

```
--x509-private-key path
```

Description

`x509-private-key` specifies the path to the private-key. The key should be in PEM format.

If it is not configured, the server instance does not load the private key and the server-side certificates.

Parameters

`path`

Path to the PEM formatted private key file. Relative and absolute paths are valid.

Examples

The instance loads the private key from `private_key.pem` which is located at *`instance_root`*`/x509`.

```
--x509-private-key ./x509/private_key.pem
```

x509-use-crl

Use the certificate revocation list

Syntax

```
--x509-use-crl
```

Description

`x509-use-crl` specifies that the server instance uses the certificate revocation list. By default, instances do not use any certificate revocation lists. In case, the CRLs in the CA store are ignored.

If `x509-use-crl` is added, the CRLs are loaded and participate in the client certificate verification. If the CRL has expired, the SSL handshake is rejected.

Examples

The instance uses certificate revocation list when authenticating clients.

```
--x509-use-crl
```

x509-use-system-store

Use the CA store provided by the system

Syntax

```
--x509-use-system-store
```

Description

`x509-use-system-store` specifies that the server instance uses the system provided CA store. By default, the server uses the file `/etc/ssl/certs/ca-certificates.crt` as trusted CA store and searches for trusted certificates under the folder `/etc/ssl/certs`. You can override these locations by setting the environment variables `SSL_CERT_FILE` and `SSL_CERT_DIR`.

Examples

The instance uses the system CA store.

```
--x509-use-system-store
```