Integrating Symphony™ with MoSes™ User Guide

Platform Symphony
Version 5.1
April 2011

Platform Computing

Contents

About the Symphony—MoSes integration

About this document

The Platform Symphony™ software (“Symphony”) works with MoSes™ from the Tillinghast business of Towers Perrin.

This document provides instructions for installing the Symphony—MoSes integration package on the machine on which the MoSes master is installed and testing the installation. Once integrated, MoSes can act as a client to Symphony and submit work to run on the grid.

License agreement

Usage of this integration software is contingent upon acceptance of the terms and conditions of the Platform Computing Corporate Software License Agreement (the "Clickwrap Agreement") accompanying the Symphony software.

Limitations

Summary	Description	Avoidance
The OpenSSL dynamic library (libeay32.dll) in the Mose6.4 package is not compatible with the same library in the Symphony 5.1 package.	Windows uses the following search path to locate a DLL that is mentioned at: http://msdn.microsoft.com/en-us/library/7d83bc18(VS.80).aspx 1. The directory where the executable module for the current process is located. 2. The current directory. 3. The Windows system directory. The GetSystemDirectory function retrieves the path of this directory. 4. The Windows directory. The GetWindowsDirectory function retrieves the path of this directory. 5. The directories listed in the PATH environment variable.	As a workaround, when this issue happens, you can copy the library to the directory where the executable is located; for instance: Copy the %SOAM_HOME%\5.1\win32-vc7\lib\libeay32.dll to the following directories: %SOAM_HOME%\5.1\win32-vc7\bin %SOAM_HOME%\5.1\win32-vc7\etc %SOAM_HOME%\..\1.2.5\bin %SOAM_HOME%\..\1.2.5\etc

About the integration

The goal of the Platform Symphony—MoSes integration is to increase MoSes performance and scalability by distributing MoSes work units to Symphony to run in parallel.

The MoSes—Symphony integration allows MoSes to submit workload to the grid, and allows MoSes workers to be automatically started and managed by Symphony.

MoSes acts as a client application to submit work requests to the grid. The integration layer converts MoSes requests to Symphony tasks. Symphony then schedules workload, allocates resources, starts up

MoSes workers on Symphony compute hosts, monitors the running of MoSes workers, and returns the location of MoSes worker output data to MoSes.

The integration is composed of the following:

- Integration binaries
- Symphony application profile `Moses6App.xml` that references the Symphony/MoSes wrapper service —`MosesService.exe`
- Symphony/MoSes wrapper service package —`MoSes6Service.zip`
- Test utility to test that the integration works—`MosesPing60.exe` and associated configuration file `<SYM_MOSES_HOME>\conf\symphonyMoses.cfg`.
- Configuration file for the Integration—`symphonyMoses.cfg`

Supported systems

MoSes version

MoSes 6.4.1.1

Symphony version

Symphony 5.1

Supported operating systems

Supported operating systems are Windows operating systems supported by MoSes for the MoSes master and worker nodes, and by Platform Symphony 5.1 for management and compute hosts.

How MoSes works with Symphony

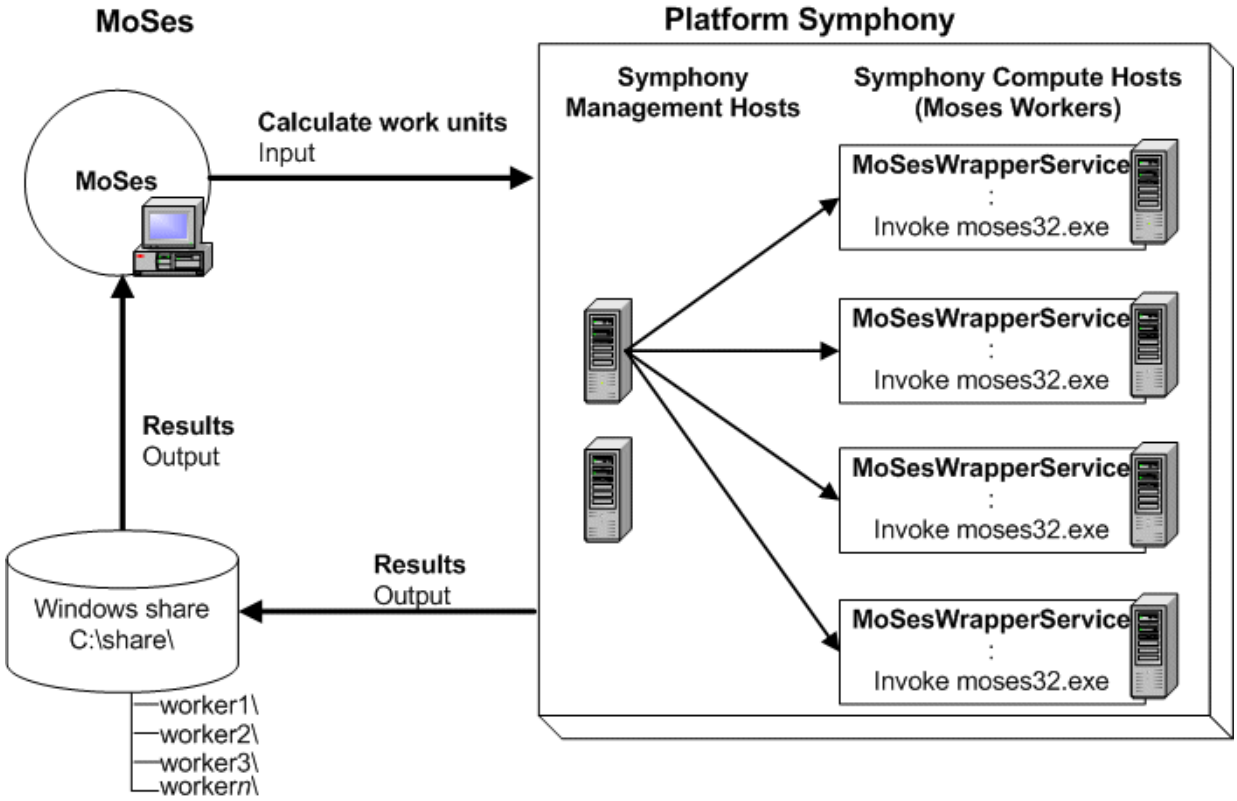
MoSes acts as a Symphony client that submits work requests to the grid. MoSes work units are submitted as Symphony tasks.

When work is submitted to Symphony, Symphony invokes the Symphony-Moses wrapper service, which does the following:

1. Creates a worker directory in the shared directory for each Symphony task
2. Parses the input provided by MoSes to retrieve running parameters for the MoSes worker, moses32.exe
3. Starts moses32.exe on each compute host
4. Writes result files in the shared directory for each Symphony task
5. Returns the location of the calculation results to MoSes

When all tasks are finished, MoSes starts aggregating the results and presents the results within the MoSes environment.

Should a task fail, Symphony automatically reruns the task. Note that Symphony reports only successful task runs to MoSes.



Symphony concepts

Cluster concepts

Symphony master

The bulk of the intelligence in the system resides on the Symphony master, which receives requests from clients and interacts with the underlying nodes to gather resource information.

Symphony hosts contain the local information collection and execution agents taking instructions from the Symphony master.

Resources

Resources are physical and logical entities that are used by applications in order to run. A resource of a particular type is associated with attributes. For example, a host has attributes of memory, CPU utilization, operating system type, etc. Platform Symphony deals with resource allocation at the granularity of physical hosts, logical sub-divisions of the physical host known as cpu slots, software license features, and includes an extensible resource model to cover storage space, network bandwidth, or data sets as resources whose use is controlled under policies.

Consumers

A consumer is a generalized notion of something that uses a resource. A consumer may be an individual user, user group, application, project, department, or an entire company. Consumers are organized hierarchically to model the nature of an organization that wants to access compute resources.

Workload-related concepts

Application

A service-oriented application is a type of application software, where the business logic is encapsulated in one or multiple software programs called services that are separated from its client logic.

Application profile

The application profile is an XML file that defines the properties of a Symphony application, including the name of the service that performs the calculation and the scheduling parameters to apply.

The application profile contains runtime parameters for workload, service, and the middleware that define how Symphony runs workload. An application profile provides flexibility to dynamically change application parameters without requiring you to change your application code and rebuild the application.

An application profile is associated with an application. An application is associated with one consumer. You must register the application profile of every application you want Symphony to manage.

Symphony client application

A program or executable that needs work done through a service. Requests are submitted via an API to the service.

Symphony service

A service is a self-contained business function that accepts one or more requests and returns one or more responses through a well-defined, standard interface.

The service performs work for a client program. It is a component capable of performing a task, and is identified by a name. Platform Symphony runs services on hosts in the cluster.

The Symphony service is the part of your application that does the actual calculation. The service encapsulates business logic.

Session

A group of tasks that share common characteristics, such as data.

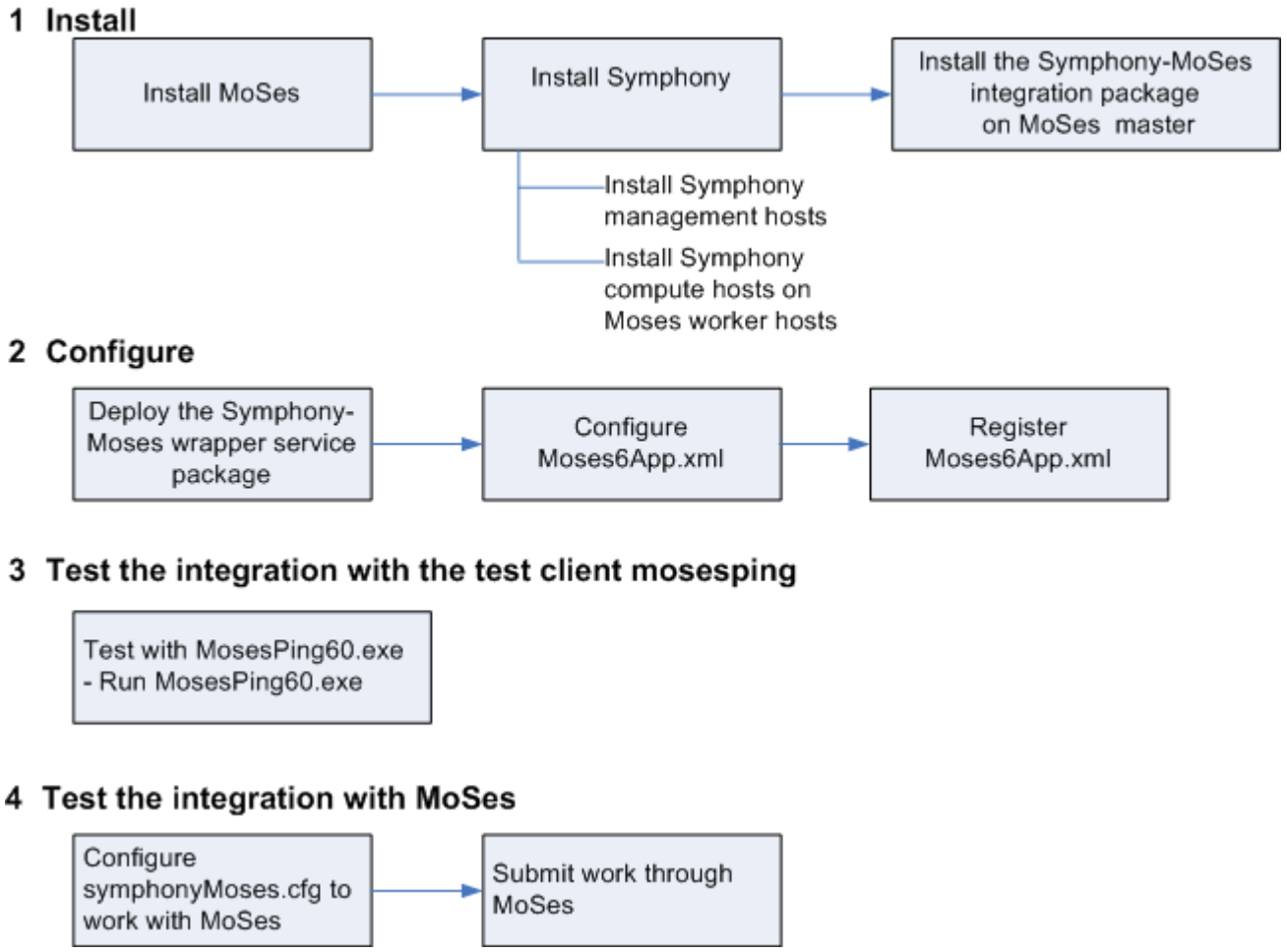
Connection

The connection on which a session is created provides a conduit for the tasks.

Task

A task is the unit of work that runs on each individual host when Symphony workload is running. The task consists of a message request (input) and, when completed by a service, a response (output).

Installation overview



Installation prerequisites

Ensure you have Local Administrator privileges to install the package

Ensure you have a Local Administrator account on the machine on which MoSes master is installed.

Know the Symphony consumer

The instructions in this document assume that you will use the /SampleApplications/SOASamples consumer to deploy the Symphony—MoSes wrapper service.

If you want to deploy the service to a different consumer, create the consumer in Symphony before installing the Symphony—MoSes integration package.

Note user accounts for the Platform Management Console

By default, when you first install Symphony, you use the following cluster administrator account in the Platform Management Console.

Administrator

- User name: Admin
- Password: Admin

If you have different user accounts/passwords in your Platform Management Console, make a note of them.

Download the packages

- Download the Symphony 5.1 DE msi package
- Download the Symphony—MoSes integration msi package

Install MoSes

Install MoSes and ensure it is working properly.

Refer to the MoSes documentation for more details.

Install Symphony

Install Symphony and ensure it is working properly.

1. Obtain a Symphony license file from Platform Computing.
2. Ensure you install the Symphony compute host packages on MoSes worker hosts.
3. Install Symphony and ensure it is working properly.

Note:

Ensure the WebServiceGateway service in EGO is running. If the port of the WebServiceGateway service is changed, change the parameter 'WSG_PORT' in SymphonyMoses.cfg to the corresponding port.

Refer to *Overview: Installing Your Platform Symphony Cluster* (install_grid_sym.pdf) in the Symphony Knowledge Center for instructions.

Install Symphony DE on the MoSes master

Symphony DE is required on the MoSes master since the master will act as a client to Symphony.

Note:

If the Symphony-Moses integration is installed on the Symphony client host rather than on a Symphony management host and compute host, and the cluster has two or more master management hosts, reconfigure the parameter 'EGO_MASTER_LIST' in ego.conf accordingly.

1. Install Symphony DE on the MoSes master.

Set up a Windows share

Set up a Windows share that can be accessed by both Symphony and MoSes.

Symphony writes calculation results to this directory, and MoSes retrieves results from this directory.

A shared directory is also required for the MoSes standard master/worker mode. Use the same directory for the integration.

1. On the file server, create the shared directory.

   For example, `c:\moses_share`
2. Create one worker directory under it.

   For example, c:\moses_share\workers

   MoSes workers will create subdirectories under it.
3. On the MoSes master and worker hosts, create a network drive mapping to the shared directory.
4. Ensure the following user accounts can read and write to the shared directory:
   - Symphony consumer OS user account
   - User account that is running MoSes on the MoSes master

Install the Symphony—MoSes integration package

See *Installation prerequisites* on page 11

1. On the MoSes master, install the Symphony 5.1 Developer Edition.

   You need this package to be able to connect to Symphony from a machine that is not in the cluster.
2. Copy the Symphony—MoSes integration msi package to the MoSes master.
3. Install the Symphony—MoSes integration msi package on the MoSes master.

Configure the Symphony—MoSes application

To distribute MoSes work units to the grid, you need to deploy the Symphony—MoSes service. To use it, you need to have an application associated with your service.

The installation package contains an application profile to register in your cluster.

1. Modify the MoSes application profile `Moses6App.xml` located in `<SYM_MOSES_HOME>\service`.
   a) In the Consumer section, if you have created a different consumer and/or resource group for the Symphony—MoSes application, change the consumerId and/or resourceGroupName, respectively.

```
<Consumer applicationName="MosesApp"
 consumerId="/SampleApplications/SOASamples"                resResq=""
policy="R_Proportion" resourceGroupName="ComputeHosts"..."/>
```

   b) In the SSM and SIM sections:

      1. Add SOAM_HOME to indicate the `soam` subdirectory under the directory in which Symphony is installed. For example, if Symphony 5.1 is installed in c:\Symphony, specify `c:\Symphony\soam` as your SOAM_HOME.

```
...
    <SOAM>
      <SSM resReq="NTX86" resourceGroupName="ManagementHosts">
            <osTypes>
                <osType name="NTX86"
                   ...
                  <env name="SOAM_HOME">C:\symphony\soam</
env>                              ...                 </osType>             </
osTypes>           ...      </SSM>      <SIM>           <osTypes>
<osType name="NTX86"                     ...                <env
name="SOAM_HOME">C:\symphony\soam</env>            ...                  </
osType>             </osTypes>        ...
```

      2. Delete the following line:

```
<env name="VERSION_NUM">3.1</env>
```

   c) In the Service section, change the fileNamePattern for the service log:

```
<osTypes>
 <osType name="all" startCmd="${SOAM_DEPLOY_DIR}
\MoSesService.exe" fileNamePattern="mosesint.service"
 logDirectory="${SOAM_HOME}/logs"
 workDir="${SOAM_HOME}/work">
  <env name="MOSES_INT_LOGLEVEL">INFO</env>
 </osType>
</osTypes>
```

   d) In the Service section, add the Control section:

```
...
</osTypes>
 <Control>
   <Method name="Invoke" >
     <Timeout duration="0" actionOnSI="restartService" actionOnWorkload="retry" /
>
     <Exit actionOnSI="restartService" actionOnWorkload="retry" />
     <Return controlCode="0" actionOnSI="keepAlive" actionOnWorkload="succeed"/>
     <Exception type="failure" controlCode="0" actionOnSI="keepAlive"
     actionOnWorkload="retry"/>
     <Exception type="fatal" controlCode="0" actionOnSI="keepAlive"
     actionOnWorkload="fail"/>
   </Method>
```

```
</Control>
</Service>
...
```

Deploy and register the Symphony—MoSes application

1. Log on to the MoSes master.
2. Go to the directory in which the Symphony—Moses package is located in: `<SYM_MOSES_HOME>\service`.
3. Deploy the `MoSes6Service.zip` service package with the `soamdeploy` command.

   Important:

   Your package name must match the package name specified in the application profile, and the consumer name must be the same as specified in your application profile.

   soamdeploy add Moses6Service -p MoSes6Service.zip -c /SampleApplications/SOASamples
4. Check the list of deployed services with the `soamdeploy view` command:

   soamdeploy view -c /SampleApplications/SOASamples

   You should be able to see the `Moses6Service` package
5. Register the application with the `soamreg` command:

   For example:

   soamreg <SYM_MOSES_HOME>\service\MoSes6App.xml

   The application is registered and enabled.
6. Check the list of applications:

   soamview app

   You should be able to see MoSesApp under the `\SampleApplications\SOASamples` consumer.

   Check that the application is enabled. If it is not, use `soamcontrol app enable` to enable the application.
7. Restart the MoSes master.

Install MoSes workers on compute hosts

The MoSes worker can be installed on compute hosts by using the MoSes Worker Manager; refer to the MoSes Version 6 Worker Manager User Guide on the MoSes installation CD for details.

Test the Symphony-MoSes integration

To test the integration, you will run the sample `MosesPing60.exe` client application. This client application sends workload to a cluster to check that the cluster is working.

1. Share the `<SYM_MOSES_HOME>\samples\sample_share` directory. Provide full access to everyone.

   Note:

   Ensure that the Symphony-Moses6 integration layer is installed and the SYM_MOSES_HOME environment variable is set.

2. Run the client application.

   The MosesPing60 application sends 4 tasks to Symphony. Symphony then invokes the Symphony—Moses service that invokes `moses32.exe` to retrieve inputs for calculation.

   <SYM_MOSES_HOME>\bin>MosesPing60.exe

   You should see output similar to the following:

```
SymPing60 will send 4 tasks to Symphony and will check the task status every 10 seconds.
Successfuly created job <103>
Sent task id <1>
Sent task id <2>
Sent task id <3>
Sent task id <4>
Task info id <1> submit time<7/6/2007 4:29:08 PM> elapsedTime<0> state<9> priority<0> skipCount<0>
workerName<hb04b12.lsf.platform.com>
Task info id <2> submit time<7/6/2007 4:29:08 PM> elapsedTime<0> state<9> priority<0> skipCount<0>
workerName<hb04b12.lsf.platform.com>
Task info id <3> submit time<7/6/2007 4:29:08 PM> elapsedTime<0> state<9> priority<0> skipCount<0>
workerName<hb04b09.lsf.platform.com>
Task info id <4> submit time<7/6/2007 4:29:08 PM> elapsedTime<0> state<9> priority<0> skipCount<0>
workerName<hb04b09.lsf.platform.com>
...
Task info id <1> submit time<7/6/2007 4:29:08 PM> elapsedTime<38> state<11> priority<0>
skipCount<0> workerName<hb04b12.lsf.platform.com>
Task info id <2> submit time<7/6/2007 4:29:08 PM> elapsedTime<42> state<11> priority<0>
skipCount<0> workerName<hb04b12.lsf.platform.com>
Task info id <3> submit time<7/6/2007 4:29:08 PM> elapsedTime<48> state<11> priority<0>
skipCount<0> workerName<hb04b09.lsf.platform.com>
Task info id <4> submit time<7/6/2007 4:29:08 PM> elapsedTime<53> state<11> priority<0>
skipCount<0> workerName<hb04b09.lsf.platform.com>
Get result for task id <1> worker dir<\\IB05B10\sample_share\rootWorkerDir\worker0> worker name<>
Get result for task id <2> worker dir<\\IB05B10\sample_share\rootWorkerDir\worker1> worker name<>
Get result for task id <3> worker dir<\\IB05B10\sample_share\rootWorkerDir\worker2> worker name<>
Get result for task id <4> worker dir<\\IB05B10\sample_share\rootWorkerDir\worker3> worker name<>
```

Configure MoSes to submit work to the cluster

MoSes has a client configuration file on the MoSes master so that MoSes can connect to Symphony. The Symphony—MoSes integration also requires the use of a specific DLL to configure Symphony as the GRID integration layer provider.

Edit the Symphony client configuration file

1. Open the configuration file `<SYM_MOSES_HOME>\conf\symphonyMoses.cfg`.
2. Edit values in the file to correspond to actual values for your application.

   Generally, you only need to change the USER, PASS, and MAP_DRIVES values.

   For more details on these parameters, see *Client Configuration File Reference* on page 21
3. To test your configuration, run MoSes and submit workload.

   Use the popup dialog or the Platform Management Console to monitor progress.

Configure Symphony as the GRID integration layer provider

1. Open the MoSes GUI.
2. Select Setup > Options > Master/Worker > Integration Layer Provider
3. Specify `%SYM_MOSES_HOME%\bin\Sym4Moses60CPP.dll`

Troubleshooting

Error messages

Should errors occur, check the log files; see Log files and log levels.

Blocked hosts

Worker hosts that fail to run `moses32.exe` are added to the blocked host list so no other tasks are dispatched to these hosts. To remove a host from the blocked host list, use the Platform Management Console in Symphony Workload >Monitor Workload, or use the command `egosh alloc unblock`.

Log files and log levels

Log files are written in `%SOAM_HOME%\logs`.

- On the compute hosts you can find the log `%SOAM_HOME%\logs\mosesint.service.log`.
- On the MoSes master you can find the log `%SOAM_HOME%\logs\mosesint.client.log`.

On the MoSes master, DEBUG level information is written to `C:\MoSes\api.hostname.log` by default. To change the log level, set the environment variable MOSES_INT_LOGLEVEL=INFO or ALL on the MoSes master.

To change log levels for the integration layer on compute hosts, use the parameter MOSES_INT_LOGLEVEL in the application profile associated with the Symphony—MoSes wrapper service.

Client Configuration File Reference

The client configuration file contains parameters for MoSes to connect to the cluster.

All parameters in the configuration file are mandatory.

Create a configuration file for every client application that connects to the MoSes master and submits work to MoSes, and the cluster.

Sample File

This file is `<SYM_MOSES_HOME>\conf\symphonyMoses.cfg`.

```
#*********************************************************
# Symphony client application configuration file.
#
# There must be one configuration file per client application.
#
# Copy this configuration file to the machine on which
# MoSes is installed.
#*********************************************************
# The user and password should be set according to the configuration of Symphony
USER = Guest
PASS = Guest
# How long Symphony will allow moses32.exe process to be idle (use 0% CPU) before it
# kills the process
# This is to prevent moses32.exe hanging forever
CHECK_IDLE_TIME = 1500
# The following parameters should be set according to the application profile.
APPLICATION_NAME = Moses6App
SESSION_NAME = MosesSession
SESSION_TYPE = ShortRunningTasks
# This parameter is used to list the locally mapped drives on the master host to be
# replicated on the compute
# hosts. This parameter is mandatory even if the drives are already mapped on the
# compute hosts.
# MAP_DRIVES=X
# The following parameter should be set according to wsg.conf.
WSG_PORT=9090
```

Parameters

USER

Symphony user account to connect to the cluster. This is the same account you use to connect to the Platform Management Console.

The user account must have been defined through the Platform Management Console.

Syntax

USER=*user_name*

Required/Optional

Required

Default value

There is no default. You need to create a user account with consumer administrator privileges and specify this account.

PASS

Password that corresponds to the user account specified with the USER parameter.

Syntax

PASS=*password*

Required/Optional

Required

Default value

There is no default. You need to specify a password.

CHECK_IDLE_TIME

Time, in seconds, that the `moses32.exe` can be idle before it is terminated. Idle time is based on CPU utilization. This parameter should be set long enough to allow a task to complete.

Syntax

CHECK_IDLE_TIME=*seconds*

Optional

Optional

Default value

1500 seconds

SESSION_NAME

Name assigned to the session when created through the client application.

Syntax

SESSION_NAME=*name*

Required/Optional

Required

Default value

MosesSession

SESSION_TYPE

Session type as specified in the `MoSesApp.xml` application profile.

Syntax

SESSION_TYPE=*type*

Required/Optional

Required

Default value

ShortRunningTasks

APPLICATION_NAME

Application name as indicated in the application profile. The application name is used to identify which consumer is associated with the application, through the application profile.

Syntax

APPLICATION_NAME=*name*

Required/Optional

Required

Default value

MosesApp

MAP_DRIVES

List of locally mapped drives on the master host to be replicated on the compute hosts. This parameter is mandatory even if the drives are already mapped on the compute hosts. Due to the nature of the Symphony services in some environments, the service will not be able to pick up the mapped network drives. This feature will automatically map the drives on the compute hosts in the same way they are mapped on the master host.

Syntax

MAP_DRIVES=*drive_letters*

For example, if the user wants to replicate the mappings of drive X, the parameter should be:

MAP_DRIVES=X

Note:

Multiple drive letters should not be separated by any characters.

Required/Optional

Required

WSG_PORT

Set the value of WSG_PORT to enable communication between the Moses integration and EGO. This parameter allows the user to configure the end point for the WSG port in the event that the cluster has been configured to not use the default port. When not specified, the integration will default to port "9090".

The value of WSG_PORT should be the same as the value of WSG_PORT configured in `%EGO_TOP%\kernel\conf\wsg.conf`.

Syntax

WSG_PORT=*number*

Required/Optional

Optional

Default value

9090

Default Configuration for the MoSes Application

The following are default values used for the MoSes application that is deployed in the cluster.

Should you want to change these default values, edit `MoSes6App.xml` and specify different values. For more details on application profile parameters, refer to the *Platform Symphony Reference*.

Default Values

consumerId="/SampleApplications/SOASamples"

By default, the consumer for the application is `/SampleApplications/SOASamples`. If you are using a different consumer name, you need to change this in the application profile.

SessionTypes section, <Type name="ShortRunningTasks" ... recoverable="false"...>

By default, the session type is set to nonrecoverable, which means the session cannot be recovered if the session manager is recovered.

Service section, <env name="SHARE_WORKER_DIRECTORY">...</env>

When this parameter is not set, the shared directory location is automatically passed by MoSes to Symphony. In addition, the system checks for the existence of the shared MoSes directory when the service makes a createSession() call.

When this parameter is set, you need to specify the location of the shared directory.

Specify the Windows directory used by MoSes as a UNC path. This is the same shared directory used for MoSes standard master/worker mode. Symphony writes calculations results to this directory, MoSes retreives results from this directory.

In addition, the system checks for the existence of the specified shared directory when the service makes an onCreateService() call.

Service section, <env name="MOSES_INT_LOGLEVEL">INFO</env>

By default, log level is set to view all messages that are logged in the information level.

You may want to change this in a production environment to log only some levels.

Note that lower log levels include higher log levels. For example, if the log level is set to WARN, messages included are WARN, ERROR, and FATAL.

Possible values are:

- FATAL
- ERROR
- WARN
- INFO
- DEBUG
- ALL

Service section, <env name="MOSES_INT_DRIVEMAP_RETRIES_MAX ">*num_of_retries*</env>

This parameter represents the number of times the service will retry a drive mapping operation if the first attempt fails. Default value is "0", i.e., no retries. This means that if you do not specify this parameter in your configuration and a drive mapping fails during a binding operation, the mapping will not be retried and the OnSessionEnter() method will issue an exception immediately.

Valid values: positive integer or 0

Service section, <env name="MOSES_INT_DRIVEMAP_RETRIES_INTERVAL ">*interval*</env>

This parameter represents the number of seconds to wait between retry attempts. Default value is "10", i.e., wait 10 seconds before the next retry.

Example:

MOSES_INT_DRIVEMAP_RETRIES_MAX="3"

MOSES_INT_DRIVEMAP_RETRIES_INTERVAL="5"

If the initial drive mapping fails, the service will wait 5 seconds and then retry the operation up to 3 times until it either succeeds or gives up after retrying for the third time. A 5 second wait will be applied between each subsequent retry of the mapping.

Valid values: positive integer or 0

Service section, <env name="MOSES_INT_ERRLEVEL_RETRIES_MAX ">*num_of_retries*</env>

This parameter represents the number of times to retry the opening and reading of the `errlevel.txt` output file from the Moses32.exe worker if the first attempt fails. Default value is "1", i.e., 1 retry. This means that if the service cannot open the `errlevel.txt` file at the end of the task invocation, the service will try one additional time to open and read the file again after a 2 second (default) delay.

Valid values: positive integer or 0

Service section, <env name="MOSES_INT_ERRLEVEL_RETRIES_INTERVAL ">*interval*</env>

This parameter represents the number of seconds to wait between retry attempts. Default value is "2", i.e., wait 2 seconds before next retry.

Example:

MOSES_INT_ERRLEVEL_RETRIES_MAX="5"

MOSES_INT_ERRLEVEL_RETRIES_INTERVAL="5"

If the initial attempt to open and read the `errlevel.txt` file fails, the service will wait 5 seconds and then retry the operation up to 5 times until it either succeeds or gives up after retrying for the fifth time. A 5 second wait will be applied between each subsequent retry to open and read the file.

Valid values: positive integer or 0

Index

U

W