Connector for Microsoft Excel User Guide

Platform Symphony™
Version 5.1
April 2011

Platform Computing

Copyright

© 1994-2011 Platform Computing Corporation

All rights reserved.

Although the information in this document has been carefully reviewed, Platform Computing Corporation ("Platform") does not warrant it to be free of errors or omissions. Platform reserves the right to make corrections, updates, revisions or changes to the information in this document.

UNLESS OTHERWISE EXPRESSLY STATED BY PLATFORM, THE PROGRAM DESCRIBED IN THIS DOCUMENT IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL PLATFORM COMPUTING BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT LIMITATION ANY LOST PROFITS, DATA, OR SAVINGS, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS PROGRAM.

We'd like to hear from you

You can help us make this document better by telling us what you think of the content, organization, and usefulness of the information. If you find an error, or just want to make a suggestion for improving this document, please address your comments to doc@platform.com.

Your comments should pertain only to Platform documentation. For product support, contact support@platform.com.

Document redistribution and translation

This document is protected by copyright and you may not redistribute or translate it into another language, in part or in whole.

Internal redistribution

You may only redistribute this document internally within your organization (for example, on an intranet) provided that you continue to check the Platform Web site for updates and update your version of the documentation. You may not make it available to your organization over the Internet.

Trademarks

® LSF is a registered trademark of Platform Computing Corporation in the United States and in other jurisdictions.

™ ACCELERATING INTELLIGENCE, PLATFORM COMPUTING, PLATFORM SYMPHONY, PLATFORM JOB SCHEDULER, PLATFORM ISF, PLATFORM ENTERPRISE GRID ORCHESTRATOR, PLATFORM EGO, and the PLATFORM and PLATFORM LSF logos are trademarks of Platform Computing Corporation in the United States and in other jurisdictions.

® UNIX is a registered trademark of The Open Group in the United States and in other jurisdictions.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

® Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Intel®, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Other products or services mentioned in this document are identified by the trademarks or service marks of their respective owners.

Third-party license agreements

http://www.platform.com/Company/third.part.license.htm

Third-party copyright notices

http://www.platform.com/Company/Third.Party.Copyright.htm

Contents

Overview

This document provides instructions for installing and configuring the Symphony—Excel integration package on a host where Symphony DE is installed. Once integrated, clients can submit Excel spreadsheets and data to Symphony to run on the grid.

About the connector for Excel

The Platform Symphony connector for Excel enables Excel to run as a service in Symphony and perform calculations in parallel on compute hosts in the cluster.

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.

Prerequisites for integrating Excel and Symphony

Excel version

- Microsoft Office 2000
- Microsoft Office 2002
- Microsoft Office 2003 SP2
- Microsoft Office 2007

Symphony version and platforms

Table 1: Symphony and Symphony DE version 5.1 on the following Windows operating systems:

Operating System	
Windows Server 2003	Windows Server 2003 Standard Edition
	Windows Server 2003 Enterprise Edition
	Windows Server 2003 R2 Standard Edition
	Windows Server 2003 R2 Enterprise Edition
Windows XP	Windows XP Professional
Windows 2000	Windows 2000 Server
	Windows 2000 Professional
Windows Vista	Windows Vista Business
Windows Server 2008	Windows Server 2008 Standard Edition

Install Excel

1. Install Excel on all compute hosts. Refer to Excel documentation for more details.
2. Ensure that all Microsoft Office components are installed on each compute host. Do not use the "install on first use" option during installation. This option causes Excel to pop up dialogs that cannot be programmatically removed.
3. Log onto all compute hosts at least once with the user account under which Symphony workload will be executed, to initialize the data associated with this user profile.
4. Set Excel macro security to Low. This prevents security dialogs from displaying. For Excel 2000, 2002, and 2003, go to Tools > Macro > Security. For Excel 2007, on the Developer tab, in the Code group, click Macro Security.

Install Symphony

1. Install Symphony DE on the client host and Symphony on the compute and management hosts. The system environment variable *SOAM_HOME* should point to the Symphony DE installation directory.

Basic Symphony 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).

Consumer

A consumer is a generalized notion of something that uses resources.

Log files and levels

The integration software uses the log4j logging framework. Log classes can be found in the log4j properties files located in the `conf` directory. Here are the most commonly-used logging levels in the log4j framework:

- ALL has the lowest possible rank and is intended to turn on all logging.
- DEBUG level designates fine-grained informational events that are most useful for debugging an application.
- ERROR level designates error events that might still allow the application to continue running.
- FATAL level designates very severe error events that will presumably lead the application to abort.
- INFO level designates informational messages that highlight the progress of the application at coarse-grained level.
- TRACE level designates finer-grained informational events than the DEBUG level.
- WARN level designates potentially harmful situations.

Connector for Excel components

Service

Located in `%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\service`. The service acts as a wrapper to Excel, and uses the information passed from the client: spreadsheet name, macro name and two parameters for the macro to invoke Excel on the compute host and execute the macro.

The VBA macro formats its result message and returns the result to the service. The service then sends the result back to the client. The service closes Excel upon completion of the task.

Note:

Output messages generated by the service are quite large, approximately 1 KB in size. Take this into account when examining performance.

The service also contains `ConnectorForExcel.dll`, which is a service DLL used to invoke Excel.

ConnectorForExcelDemo.xls

Located in `%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\samples\spreadsheets`. Actual spreadsheet that contains the business logic to perform calculations in Excel. Contains the MyMacro VBA function. The client sends this spreadsheet as common data for the ConnectorForMsExcel Service to process.The demo spreadsheet can be used as a template for modifying your own spreadsheets to work with the ConnectorForMsExcel service.

DialogSniffer.exe

Used for troubleshooting. Parameters for DialogSniffer are configured in the application profile. Detects dialog boxes that appear during the execution of a task, and writes the dialog box text to a log file. Can dismiss dialogs based on text patterns specified in the application profile so that Excel can continue calculations.

VBAMacroTest.xls

Autonomous spreadsheet used to debug a macro in your own spreadsheet. Uses `ConnectorForExcel.dll` directly as a COM-object without Symphony. Packaged with the service so that you can test spreadsheets on compute hosts if needed.

Installing the application

Perform the following steps on the client host.

1. Double-click the `SymphonyConnectorForMsExcel5.1.0_win32.msi` file.
2. Follow the screen prompts to install the package.
3. Go through the sample tutorial.

   Refer to Developing a connector for Excel client.

Developing a connector for Excel client

Goal

In this tutorial, you will learn how to run Excel as a service in Symphony and perform calculations on compute hosts in the cluster.

At a glance

Before you begin, ensure you have installed and started Symphony DE. You will do the following:

1. Review and understand the sample
2. Build the sample client and deploy the sample service
3. Run the sample client and service

Review and understand the sample

Review the C++ sample client code to learn how to link your Excel spreadsheets to Symphony.

The connector for Excel package also includes a VB client sample that connects to Symphony using the COM API; refer to the Cross-language tutorial in the Knowledge Center for more information.

Locate the code samples

Note:

%SOAM_HOME% is an environment variable that represents the Symphony DE installation directory; for example, C:\SymphonyDE.

Solution file (Visual Studio .NET)

```
%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\samples\CPP
\AsyncClient\connector_for_ms_excel_sample_<version>.sln
```

where <*version*> is the version of Visual Studio

Client

```
%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\samples\CPP
\AsyncClient\AsyncClient.cpp
```

Input, output, and data objects

```
%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\src\
```

Service

```
%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\service
\ConnectorForMsExcel.zip
```

Spreadsheet

```
%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\samples\spreadsheets
\ConnectorForExcelDemo.xls
```

Application profile

The service required to compute the input data along with additional application parameters are defined in the application profile:

```
%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\samples\CPP
\AsyncClient\connectorForExcelSampleApp.xml
```

What the sample does

The sample provides an asynchronous C++ client, Excel spreadsheet, and wrapper service for Excel. The Excel spreadsheet contains business logic implemented in the form of a macro (VBA function) that performs calculations.

When you run the sample, here is the sequence of events:

1. The client sends task input messages and common data to the ConnectorForMsExcel service. Each task input message contains:
   - workbook name (ex. `ConnectorForExcelDemo.xls`) containing the macro
   - macro name (ex. MyMacro)
   - input string (data) passed to the macro.

   The common data contains the path of the Excel spreadsheet.
2. The ConnectorForMsExcel service launches Excel, which opens the workbook and executes the VBA macro. The VBA macro processes the input data string that is sent with each task input message.
3. The macro formats the result message and passes it to the service. The service sends the message to the client, which displays it in the command prompt window. The service closes Excel upon completion of the tasks.

Step 1: Develop the spreadsheet macro

The spreadsheet contains the logic (VBA code) that performs the calculations on the input data. When we create the input message, we pass the name of the VBA macro and other data to the service.

In the ConnectorForExcelDemo.xls spreadsheet, we implement `MyMacro()` as the main function. It calls the `Initialize()` function, which parses the input data string to extract the values for <NumberIterations> and <Seed>.

The `Sim_RunFunction()` performs a cycle of calculations based on the number of iterations specified in the input data string and prints out the results in the spreadsheet.

Step 2: Initialize the client

In `AsyncClient.cpp`, when you initialize, you initialize the Symphony client infrastructure. You initialize once per client.

Important:

Initialization is required. Otherwise, API calls will fail.

```
...
      SoamFactory::initialize();
...
```

Step 3: Implement the response handler (callback) method to retrieve output messages

With an asynchronous client, when a task is completed by the service, there must be a means of communicating this status back to the client. The response handler or callback is implemented for this purpose. It is called by the middleware each time a service completes a task.

In this sample, the `OnResponse()` method is the response handler. It is a member of the MySessionCallback class that inherits from the SessionCallback class. The method accepts the TaskOutputHandle as an input argument, which is passed to the method by the middleware whenever the respective task has completed.

First, we check if there is output to retrieve. If so, get the output message from the service and print out the task ID. Extract and print the message from the task result using the `populateTaskOutput()` method.

Increment the counter that records the number of task results received. The critical section object ensures that another thread does not try to increment the counter while it is being accessed.

```
void onResponse(TaskOutputHandlePtr &output) throw()
{
    cout << "onResponse handler called" << endl;
    try
    {
        cout << "Check for success of task" << endl;
        // check for success of task
        if (true == output->isSuccessful())
        {
            // get the message returned from the service
            ExcelMessage outMsg;
            output->getMessage(&outMsg);
            // display content of reply
            cout << "Task Succeeded [" <<  output->getId() << "]" << endl;
            string message = outMsg.GetResult();
            if( message.length() > 512 )
            {
                string tmp = message.substr(0, 512);
                cout << tmp ;
                cout << "( to display the actual size " << (unsigned int)
                message.length() << " is truncated to 512 )" << endl << endl;
            }
            else
            {
                cout << outMsg.GetResult() << endl << endl;
            }
        }
        else
        {
            // get the exception associated with this task
            SoamExceptionPtr ex = output->getException();
            cout << "Task Failed : " << ex->what() << endl << endl;
        }
    }
    catch(SoamException &exception)
    {
        cout << "Exception occured in OnResponse() : " << exception.what() << endl;
    }
    // Update counter used to synchronize the controlling thread
    // with this callback object
    EnterCriticalSection(&m_criticalSection);
    ++m_tasksReceived;
    LeaveCriticalSection(&m_criticalSection);
}
```

Step 4: Connect to an application

To send data to be calculated in the form of input messages, you connect to an application.

You specify an application name, a user name, and password. The application name must match that defined in the application profile.

For Symphony DE, there is no security checking and login credentials are ignored—you can specify any user name and password. Security checking is done however, when your client application submits workload to the actual grid.

The default security callback encapsulates the callback for the user name and password.

```
...
    const char appName[]="ConnectorForExcel";
...
    // setup application authentication information using the default security
    //provider,
    DefaultSecurityCallback securityCB("Guest", "Guest");
...
    // connect to the specified application
    ConnectionPtr conPtr = SoamFactory::connect(appName, &securityCB);
...
    // retrieve and print our connection ID
    cout << "get connection ID=" << conPtr->getId() << endl << endl;
```

```
...
```

Step 5: Create a session to group tasks:

In `AsyncClient.cpp`, perform this step after you have connected to the application.

When creating an asynchronous session, you need to specify the session attributes by using the SessionCreationAttributes object. In this sample, we create a SessionCreationAttributes object called attributes and set five parameters in the object.

The first parameter is the session name. This is optional. The session name can be any descriptive name you want to assign to your session. It is for information purposes, such as in the command line interface.

The second parameter is the session type. The session type is optional. You can leave this parameter blank and system default values are used for your session.

The third parameter is the session flag, which we specify as SF_RECEIVE_ASYNC. You must specify it as shown. This indicates to Symphony that this is an asynchronous session.

The fourth parameter is the common data object containing the path to the Excel spreadsheet that you want to send to the service.

The fifth parameter is the callback object.

We pass the attributes object to the `createSession()` method, which returns a pointer to the session.

```
...
 // create call back
        MySessionCallback  myCallback;
        std::vector<std::string> fileNames;
        std::string pathToWorkBookeFile="C:\\SymphonyDE\\DE32\\3.2\\Integrations\\
        ConnectorForMsExcel\\samples\\spreadsheets\\ConnectorForExcelDemo.xls";
        fileNames.push_back(pathToWorkBookeFile);
        ExcelCommonData commonData( fileNames );
        cout << "Creating an asynchronous Session" << endl;
        // Create an asynchronous Session
        SessionCreationAttributes attributes;
        attributes.setSessionName("mySession");
        attributes.setSessionType("ShortRunningTasks");
        attributes.setSessionFlags(Session::ReceiveAsync);
        attributes.setCommonData(&commonData);
        attributes.setSessionCallback(&myCallback);
        // Create a synchronous session
        SessionPtr sesPtr = conPtr->createSession(attributes);
        cout << "    Session created" << endl;
        // retrieve and print session ID
        cout << "    Session ID:" << sesPtr->getId() << endl << endl;
...
```

Step 6: Send input data to be processed

In this step, we create 10 input messages to be processed by the service. We call the `fillMessageWithExcelData()` method. The `fillMessageWithExcelData()` method fills the inMsg object with (1) spreadsheet name, (2) macro name, and (3) data input string. When a message is sent with the `sendTaskInput()` method, a task input handle is returned. This task input handle contains the ID for the task that was created for this input message.

```
...
 int tasksToSend = 10;
        cout << "Send " << tasksToSend << " messages to Excel service" << endl;
        for (int taskCount = 0; taskCount < tasksToSend; taskCount++)
        {
            // Create a message
            ExcelMessage inMsg;
            cout << "Preparing message #" << taskCount +1 << endl;
            fillMessageWithExcelData(inMsg);

            cout << "     " << inMsg.ToString() << endl;
            cout << "Trying to submitt task..." << endl;
            // send it
            TaskInputHandlePtr input = sesPtr->sendTaskInput(&inMsg);

            // retrieve and print task ID
            cout << "      Task submitted with ID : " << input->getId() << endl << endl;
        }
...
```

```
...
void fillMessageWithExcelData( ExcelMessage &inMsg )
{
    char cWorkbookName[]="ConnectorForExcelDemo.xls";
    inMsg.SetWorkbookName( cWorkbookName );
    char cMacroName[]="MyMacro";
    inMsg.SetMacroName(cMacroName);
    char sInputString[]="<NumberIterations>1</NumberIterations><Seed>55545</Seed>";
    inMsg.SetParam(sInputString);
}
...
```

Step 7: Wait for replies

After all 10 tasks (messages) have been sent to the service, the main client execution thread must wait for all tasks to be processed before uninitializing the client API. As each task is completed by the service, the m_tasksReceived variable is incremented; refer to Step 3. The `myCallback.getReceived()` method returns the value of m_tasksReceived. If m_tasksReceived is less than the total number of tasks sent and there are no exceptions thrown, the main thread waits two seconds before checking the value of m_tasksReceived again. This cycle continues until all the tasks results are received.

```
...
cout << "Wait till all replies have been received asynchronously by our callback ..."
          << endl << endl;
        while ((myCallback.getReceived() < tasksToSend) && !myCallback.getDone())
        {
            ourSleep(2);
        }
...
```

```
...
void ourSleep(unsigned long sleepInSeconds)
{
    soam::Sleep(sleepInSeconds * 1000);
}
...
```

Step 8: Uninitialize

Always uninitialize the client API at the end of all API calls. If you do not call uninitialize, the client API will be in an undefined state and resources used by the client will be held indefinitely.

Important:

Once you uninitialize, all objects become null. For example, you can no longer create a session or send an input message.

```
...
SoamFactory::uninitialize();
...
```

Build the sample client and add the application

Build the sample client

1. Locate solution file `connector_for_ms_excel_sample_<version>.sln` in `%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\samples\CPP\AsyncClient`.
2. Load the file into Visual Studio and build it. You can find the compiled client binary in `%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\samples\CPP\output\`

Add the application

When you add an application through the DE PMC, you must use the Add Application wizard. This wizard defines a consumer location to associate with your application, deploys your service package, and registers your application. After completing the steps with the wizard, your application should be ready to use.

1. Click Symphony Workload > Configure Applications.

   The Applications page displays.
2. Select Global Actions > Add/Remove Applications.

   The Add/Remove Application page displays.
3. Select Add a new application, then click Continue.

   The Adding an Application page displays.
4. Select Use existing profile and add application wizard. Click Browse and navigate to `%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\samples\CPP\AsyncClient`.
5. Select application profile `connectorForExcelSampleApp.xml`, then click Continue.

   The Service Package location window displays.
6. Click Browse and navigate to `%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\service.` Select service package `ConnectorForMsExcel.zip`, then click Continue.

   The Confirmation window displays.
7. Review your selections, then click Confirm.

   The project wizard creates your application and registers it within Symphony.
8. Click Close.

   The application is now enabled.

Run the sample client and service

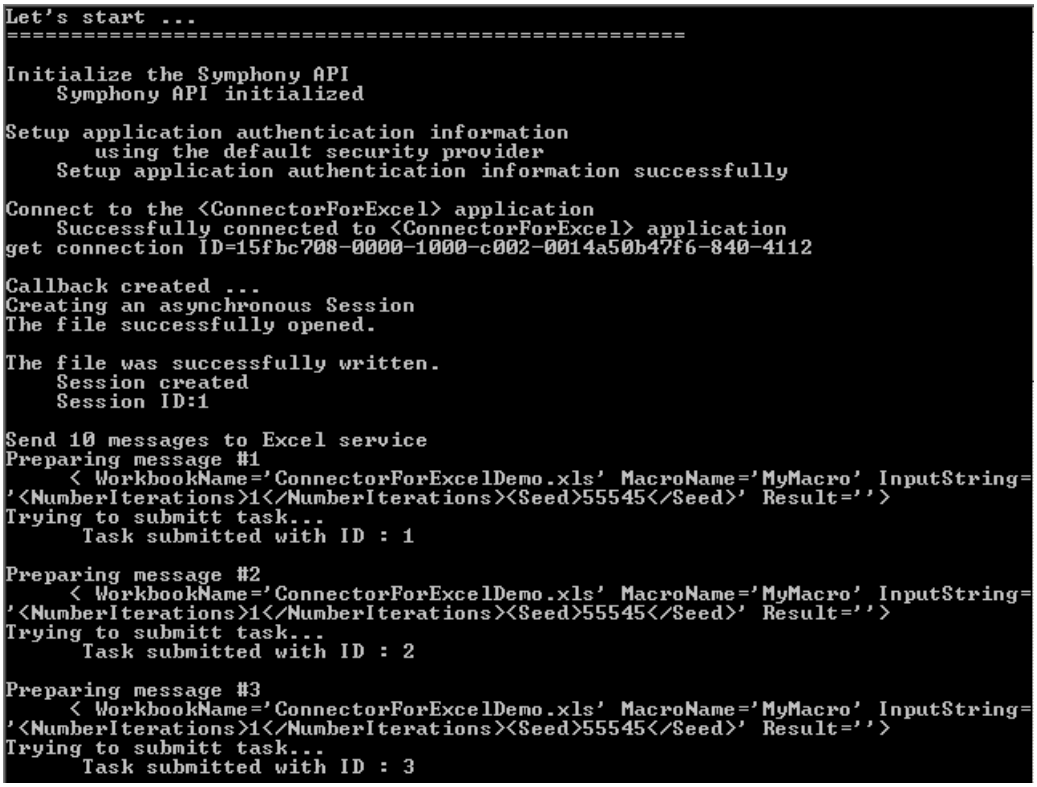
To run the service, you run the client application. The service a client application uses is specified in the application profile.

1. Go to the directory in which the client executable is located:

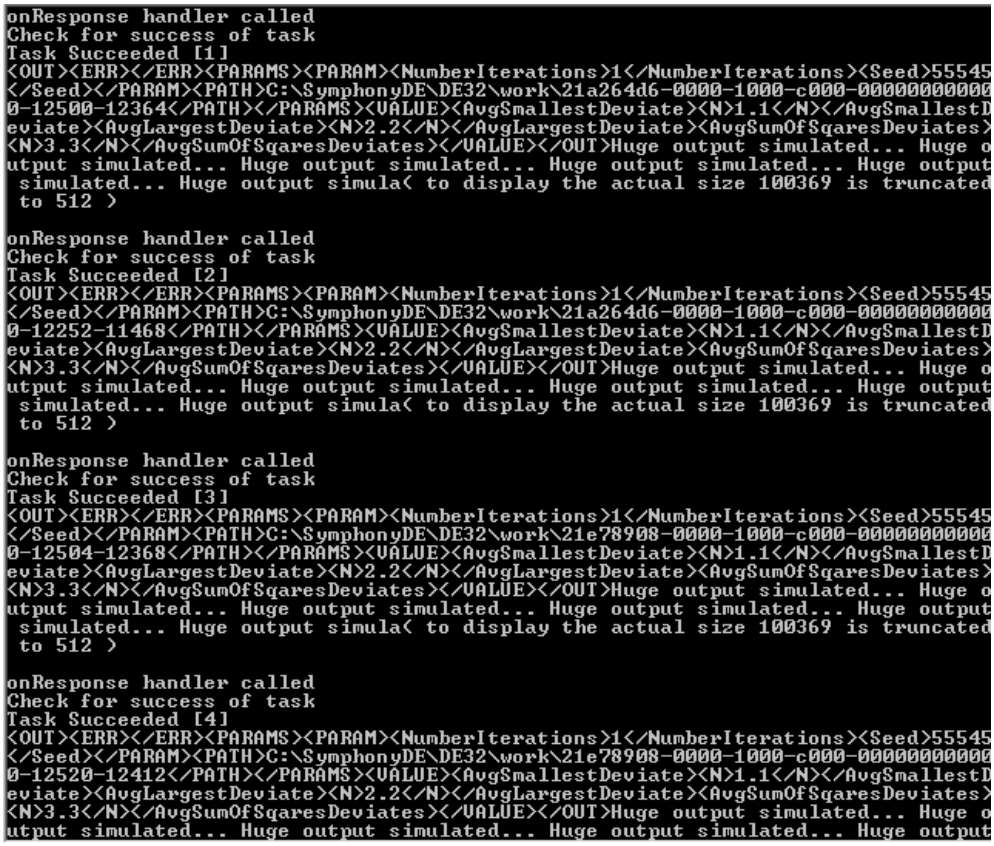
   cd %SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\samples\CPP\output
2. Run the client application:

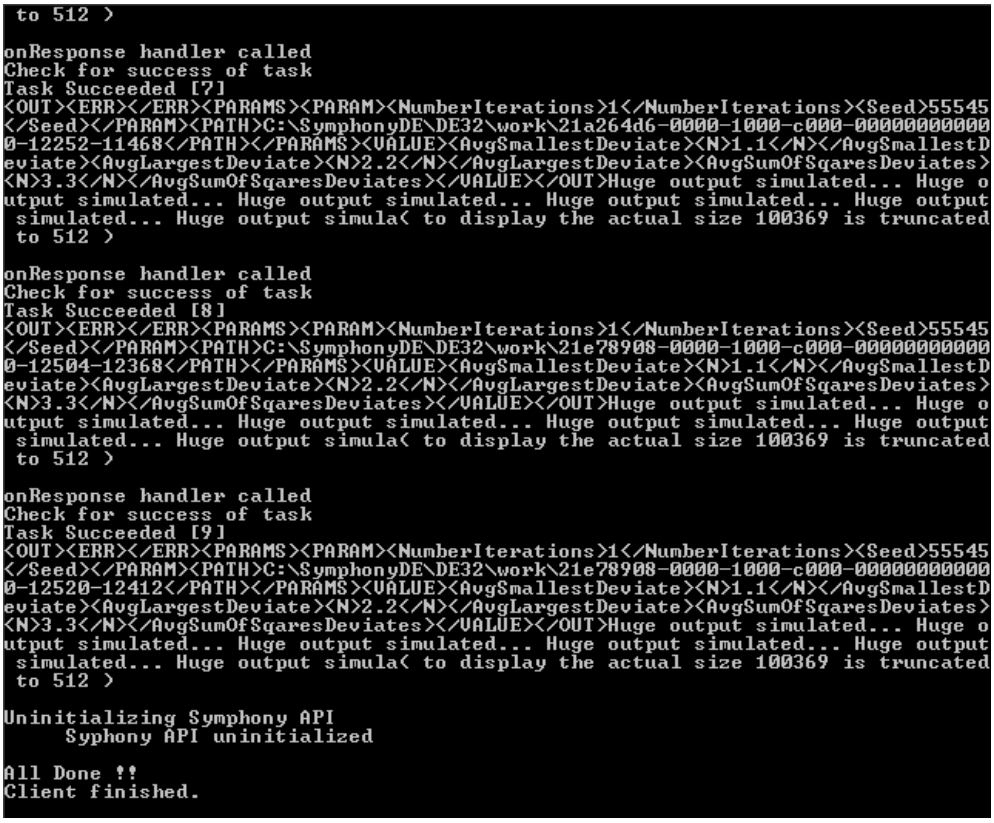
   AsyncClient.exe

   The client starts and the system starts the corresponding service. You should see the following output in the command line window as tasks are submitted to the service.


Let's start ...
==========
Initialize the Symphony API
Symphony API initialized
Setup application authentication information
using the default security provider
Setup application authentication information successfully
Connect to the <ConnectorForExcel> application
Successfully connected to <ConnectorForExcel> application
get connection ID=15fbc708-0000-1000-c002-0014a50b47f6-840-4112
Callback created ...
Creating an asynchronous Session
The file successfully opened.
The file was successfully written.
Session created
Session ID:1
Send 10 messages to Excel service
Preparing message #1
< WorkbookName='ConnectorForExcelDemo.xls' MacroName='MyMacro' InputString=
'<NumberIterations>1</NumberIterations><Seed>55545</Seed>' Result=''>
Trying to submitt task...
Task submitted with ID : 1
Preparing message #2
< WorkbookName='ConnectorForExcelDemo.xls' MacroName='MyMacro' InputString=
'<NumberIterations>1</NumberIterations><Seed>55545</Seed>' Result=''>
Trying to submitt task...
Task submitted with ID : 2
Preparing message #3
< WorkbookName='ConnectorForExcelDemo.xls' MacroName='MyMacro' InputString=
'<NumberIterations>1</NumberIterations><Seed>55545</Seed>' Result=''>
Trying to submitt task...
Task submitted with ID : 3

```
Preparing message #4
     < WorkbookName='ConnectorForExcelDemo.xls' MacroName='MyMacro' InputString=
'<NumberIterations>1</NumberIterations><Seed>55545</Seed>' Result=''>
Trying to submitt task...
      Task submitted with ID : 4

Preparing message #5
     < WorkbookName='ConnectorForExcelDemo.xls' MacroName='MyMacro' InputString=
'<NumberIterations>1</NumberIterations><Seed>55545</Seed>' Result=''>
Trying to submitt task...
      Task submitted with ID : 5

Preparing message #6
     < WorkbookName='ConnectorForExcelDemo.xls' MacroName='MyMacro' InputString=
'<NumberIterations>1</NumberIterations><Seed>55545</Seed>' Result=''>
Trying to submitt task...
      Task submitted with ID : 6

Preparing message #7
     < WorkbookName='ConnectorForExcelDemo.xls' MacroName='MyMacro' InputString=
'<NumberIterations>1</NumberIterations><Seed>55545</Seed>' Result=''>
Trying to submitt task...
      Task submitted with ID : 7

Preparing message #8
     < WorkbookName='ConnectorForExcelDemo.xls' MacroName='MyMacro' InputString=
'<NumberIterations>1</NumberIterations><Seed>55545</Seed>' Result=''>
Trying to submitt task...
      Task submitted with ID : 8

Preparing message #9
     < WorkbookName='ConnectorForExcelDemo.xls' MacroName='MyMacro' InputString=
'<NumberIterations>1</NumberIterations><Seed>55545</Seed>' Result=''>
Trying to submitt task...
      Task submitted with ID : 9

Preparing message #10
     < WorkbookName='ConnectorForExcelDemo.xls' MacroName='MyMacro' InputString=
'<NumberIterations>1</NumberIterations><Seed>55545</Seed>' Result=''>
Trying to submitt task...
      Task submitted with ID : 10

Wait till all replies have been received asynchronously by our callback ...
```


onResponse handler called
Check for success of task
Task Succeeded [1]
<OUT><ERR></ERR><PARAMS><PARAM><NumberIterations>1</NumberIterations><Seed>55545
</Seed></PARAM><PATH>C:\SymphonyDE\DE32\work\21a264d6-0000-1000-c000-00000000000
0-12500-12364</PATH></PARAMS><VALUE><AvgSmallestDeviate><N>1.1</N></AvgSmallestD
eviate><AvgLargestDeviate><N>2.2</N></AvgLargestDeviate><AvgSumOfSqaresDeviates>
<N>3.3</N></AvgSumOfSqaresDeviates></VALUE></OUT>Huge output simulated... Huge o
utput simulated... Huge output simulated... Huge output simulated... Huge output
simulated... Huge output simula(to display the actual size 100369 is truncated
to 512)
onResponse handler called
Check for success of task
Task Succeeded [2]
<OUT><ERR></ERR><PARAMS><PARAM><NumberIterations>1</NumberIterations><Seed>55545
</Seed></PARAM><PATH>C:\SymphonyDE\DE32\work\21a264d6-0000-1000-c000-00000000000
0-12252-11468</PATH></PARAMS><VALUE><AvgSmallestDeviate><N>1.1</N></AvgSmallestD
eviate><AvgLargestDeviate><N>2.2</N></AvgLargestDeviate><AvgSumOfSqaresDeviates>
<N>3.3</N></AvgSumOfSqaresDeviates></VALUE></OUT>Huge output simulated... Huge o
utput simulated... Huge output simulated... Huge output simulated... Huge output
simulated... Huge output simula(to display the actual size 100369 is truncated
to 512)
onResponse handler called
Check for success of task
Task Succeeded [3]
<OUT><ERR></ERR><PARAMS><PARAM><NumberIterations>1</NumberIterations><Seed>55545
</Seed></PARAM><PATH>C:\SymphonyDE\DE32\work\21e78908-0000-1000-c000-00000000000
0-12504-12368</PATH></PARAMS><VALUE><AvgSmallestDeviate><N>1.1</N></AvgSmallestD
eviate><AvgLargestDeviate><N>2.2</N></AvgLargestDeviate><AvgSumOfSqaresDeviates>
<N>3.3</N></AvgSumOfSqaresDeviates></VALUE></OUT>Huge output simulated... Huge o
utput simulated... Huge output simulated... Huge output simulated... Huge output
simulated... Huge output simula(to display the actual size 100369 is truncated
to 512)
onResponse handler called
Check for success of task
Task Succeeded [4]
<OUT><ERR></ERR><PARAMS><PARAM><NumberIterations>1</NumberIterations><Seed>55545
</Seed></PARAM><PATH>C:\SymphonyDE\DE32\work\21e78908-0000-1000-c000-00000000000
0-12520-12412</PATH></PARAMS><VALUE><AvgSmallestDeviate><N>1.1</N></AvgSmallestD
eviate><AvgLargestDeviate><N>2.2</N></AvgLargestDeviate><AvgSumOfSqaresDeviates>
<N>3.3</N></AvgSumOfSqaresDeviates></VALUE></OUT>Huge output simulated... Huge o
utput simulated... Huge output simulated... Huge output simulated... Huge output


to 512)
onResponse handler called
Check for success of task
Task Succeeded [7]
<OUT><ERR></ERR><PARAMS><PARAM><NumberIterations>1</NumberIterations><Seed>55545
</Seed></PARAM><PATH>C:\SymphonyDE\DE32\work\21a264d6-0000-1000-c000-000000000000
0-12252-11468</PATH></PARAMS><VALUE><AvgSmallestDeviate><N>1.1</N></AvgSmallestD
eviate><AvgLargestDeviate><N>2.2</N></AvgLargestDeviate><AvgSumOfSqaresDeviates>
<N>3.3</N></AvgSumOfSqaresDeviates></VALUE></OUT>Huge output simulated... Huge o
utput simulated... Huge output simulated... Huge output simulated... Huge output
simulated... Huge output simula(to display the actual size 100369 is truncated
to 512)
onResponse handler called
Check for success of task
Task Succeeded [8]
<OUT><ERR></ERR><PARAMS><PARAM><NumberIterations>1</NumberIterations><Seed>55545
</Seed></PARAM><PATH>C:\SymphonyDE\DE32\work\21e78908-0000-1000-c000-000000000000
0-12504-12368</PATH></PARAMS><VALUE><AvgSmallestDeviate><N>1.1</N></AvgSmallestD
eviate><AvgLargestDeviate><N>2.2</N></AvgLargestDeviate><AvgSumOfSqaresDeviates>
<N>3.3</N></AvgSumOfSqaresDeviates></VALUE></OUT>Huge output simulated... Huge o
utput simulated... Huge output simulated... Huge output simulated... Huge output
simulated... Huge output simula(to display the actual size 100369 is truncated
to 512)
onResponse handler called
Check for success of task
Task Succeeded [9]
<OUT><ERR></ERR><PARAMS><PARAM><NumberIterations>1</NumberIterations><Seed>55545
</Seed></PARAM><PATH>C:\SymphonyDE\DE32\work\21e78908-0000-1000-c000-000000000000
0-12520-12412</PATH></PARAMS><VALUE><AvgSmallestDeviate><N>1.1</N></AvgSmallestD
eviate><AvgLargestDeviate><N>2.2</N></AvgLargestDeviate><AvgSumOfSqaresDeviates>
<N>3.3</N></AvgSumOfSqaresDeviates></VALUE></OUT>Huge output simulated... Huge o
utput simulated... Huge output simulated... Huge output simulated... Huge output
simulated... Huge output simula(to display the actual size 100369 is truncated
to 512)
Uninitializing Symphony API
Syphony API uninitialized
All Done !!
Client finished.

Configuring an application

Configure custom application profile

1. Copy `%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\samples\CPP\AsyncClient\connectorForExcelSampleApp.xml` and create a new file.
2. Change the consumer section to indicate your application name and consumer:

```
...
 <Consumer applicationName="ConnectorForExcel"
consumerId="/SampleApplications/SOASamples"
.../>
...
```

3. Consider and decide which of the following parameters to change, if necessary, for your application.

```
...<env name="START_SNIFFER">false</env>
<env name="FATAL_PATTERNS">Compile error</env>
<env name="NON_FATAL_PATTERNS">The Office Assistant
   could not be started</env>
<env name="DISSMISS_DLG_WITH_PATTERNS">Simulated error
   happened</env>
<env name="FATAL_TIMEOUT">30</env>
<env name="NON_FATAL_TIMEOUT">40</env>
<env name="APP_DEPLOY_DIR">${SOAM_DEPLOY_DIR}</env>
...
```

Configure logging

1. Make a backup copy of the `api.log4j.properties` file in `%SOAM_HOME%\conf`.
2. Copy the `%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\conf\api.log4j.properties` file to `%SOAM_HOME%\conf`.

Testing and debugging spreadsheets

Test the demo spreadsheet

Before using your spreadsheet with Symphony, debug it with `VBAMacroTest.xls`. The macro in this spreadsheet performs the exact same steps as the ConnectorForExcel service.

1. Register `ConnectorForExcel.dll`.

   regsvr32 ConnectorForExcel.dll

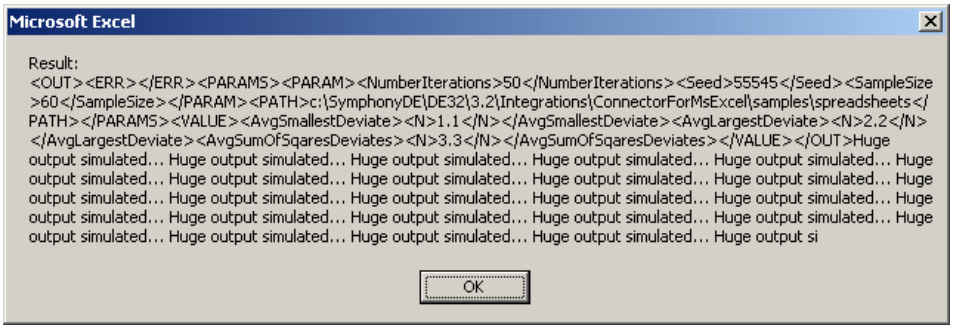
2. In Excel, open `%SOAM_HOME%\5.1\Integrations\ConnectorForMsExcel\samples\spreadsheets\VBAMacroTest.xls`.
3. Ensure the `ConnectorForExcel.dll` is referenced. The VBA code in `VBAMacroTest.xls` requires the `ConnectorForExcel.dll` COM object to be properly referenced because it attempts to create an instance of that COM object.
   a) Select Tools > Macro > Visual Basic Editor.
   b) In the Visual Basic Editor window, select Tools > References.

      The VBAProject dialog box opens.
   c) In the dialog, select the "ConnectorForMSExcel 1.0 Type Library" to place a checkmark in the checkbox.
4. Test that `VBAMacroTest.xls` works with the demo spreadsheet.
   a) In `VBAMacroTest.xls`, specify parameters to test the spreadsheet:

      1. Name of spreadsheet to test: `ConnectorForExcelDemo.xls`
      2. Absolute path to the directory that contains the spreadsheet to test:

         For example: *`%SOAM_HOME%`*`\5.1\Integrations\ConnectorForMsExcel\samples\spreadsheets`
   b) Click the button Excecute Macro.

      You should see the following output:



Test your spreadsheet

1. Open `VBAMacroTest.xls` and specify parameters for your spreadsheet:

   1. Name of spreadsheet to test: Specify the name of your Excel spreadsheet.

2. Name of VBA macro to test: Indicate the macro to execute.
3. Fill in the remaining fields as required.

Debug your spreadsheet

1. In `VBAMacroTest.xls`, use the VBA debugger to stop in the Sub CommandButton1_Click() line, and check step-by-step what `ConnectorForExcel.dll` returns in the following statements:
   - result_start = TestExcelRunnerDemo.StartExcel(pid)
   - result = TestExcelRunnerDemo.ExecuteMacro(SheetName, MacroName, Param, PathPrefix)
   - result_quit = TestExcelRunnerDemo.QuitExcel()
2. Examine the dialog box messages raised during execution of your macro and eliminate their causes.
3. Consider your current memory threshold setting in Excel, and ensure it is set at a level that will allow macro execution. See Troubleshooting section for details.

Troubleshooting

This section provides troubleshooting tips in response to error messages that may appear when running your application.

"Out of memory" dialog in Excel

Reason:

An "Out of Memory" message may be reported if there is insufficient memory available to complete each process, even if it seems that there is sufficient memory available in the system as a whole. In this case, the Dialog Sniffer log file may contain one or both of the following messages:

I_n_s_p_e_c_t_W_i_n_d_o_w_H_o_o_k__i_s__i_n_v_o_k_e_d__i_n_ r_e_s_p_o_n_s_e__t_o_ W_M__I_N_I_T_D_I_A_L_O_G_. Tue Oct 19 12:17:47 2004 E_n_t_e_r_e_d__I_s_A_M_e_s_s_a_g_e_ B_o_x__f_u_n_c_t_i_o_n_._._.
T_h_e__o_p_e_n_e_d__w_i_n_d_o_w_'_s__c_l_a_s_s__n_a_m_e__i_s_:__<#32770>
T_h_e__o_p_e_n_e_d__w_i_n_d_o_w_'_s__t_e_x_t__i_s_:__<Microsoft Visual Basic>
S_t_a_t_i_c__t_e_x_t__i_s_:__<Microsoft Visual Basic>
D_e_s_t_r_o_y__p_a_t_t_e_r_n_s__r_e_t_r_i_e_v_e_d_. B_e_g_i_n__t_o_ _i_n_s_p_e_c_t__t_h_e__w_i_n_d_o_w_. C_l_a_s_s__N_a_m_e_:__<Button>
B_u_t_t_o_n__c_o_n_t_a_i_n_s__t_h_i_s__t_e_x_t:__<OK> B_e_g_i_n__t_o_ _i_n_s_p_e_c_t__t_h_e__w_i_n_d_o_w_. C_l_a_s_s__N_a_m_e_:__<Button>
B_u_t_t_o_n__c_o_n_t_a_i_n_s__t_h_i_s__t_e_x_t:__<Help> B_e_g_i_n__t_o_ _i_n_s_p_e_c_t__t_h_e__w_i_n_d_o_w_. C_l_a_s_s__N_a_m_e_:__<Static>
_G_e_t_W_i_n_d_o_w_T_e_x_t__r_e_t_u_r_n_s__t_h_e__e_r_r_o_r_:__ <Cannot create a file when that file already exists. > B_e_g_i_n__t_o_ _i_n_s_p_e_c_t__t_h_e__w_i_n_d_o_w_. C_l_a_s_s__N_a_m_e_:__<Static>
S_t_a_t_i_c__t_e_x_t__i_s_:__<Out of memory>

I_n_s_p_e_c_t_W_i_n_d_o_w_H_o_o_k__i_s__i_n_v_o_k_e_d__i_n_ r_e_s_p_o_n_s_e__t_o_ W_M__I_N_I_T_D_I_A_L_O_G_. Tue Oct 19 12:24:25 2004 E_n_t_e_r_e_d__I_s_A_M_e_s_s_a_g_e_ B_o_x__f_u_n_c_t_i_o_n_._._.
T_h_e__o_p_e_n_e_d__w_i_n_d_o_w_'_s__c_l_a_s_s__n_a_m_e__i_s_:__<#32770>
T_h_e__o_p_e_n_e_d__w_i_n_d_o_w_'_s__t_e_x_t__i_s_:__ <Microsoft Excel>
S_t_a_t_i_c__t_e_x_t__i_s_:__<Microsoft Excel> D_e_s_t_r_o_y__p_a_t_t_e_r_n_s__r_e_t_r_i_e_v_e_d_. B_e_g_i_n__t_o_ _i_n_s_p_e_c_t__t_h_e__w_i_n_d_o_w_. C_l_a_s_s__N_a_m_e_:__<Button>
B_u_t_t_o_n__c_o_n_t_a_i_n_s__t_h_i_s__t_e_x_t:__<OK> B_e_g_i_n__t_o_ _i_n_s_p_e_c_t__t_h_e__w_i_n_d_o_w_. C_l_a_s_s__N_a_m_e_:__<Static>
_G_e_t_W_i_n_d_o_w_T_e_x_t__r_e_t_u_r_n_s__t_h_e__e_r_r_o_r_:__ <Cannot create a file when that file already exists. > B_e_g_i_n__t_o_ _i_n_s_p_e_c_t__t_h_e__w_i_n_d_o_w_. C_l_a_s_s__N_a_m_e_:__<Static>
S_t_a_t_i_c__t_e_x_t__i_s_:__<Not enough memory to run Microsoft Excel.>

Solution:

1. Increase the memory size per process.
2. Increase the desktop heap value as follows:
    1. From the Start menu, select Run.
    2. Enter **regedit** to invoke the registry editor.
    3. Go to `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystems\`.

4. Double-click the Windows parameter to display the value data.

   For example:

   ```
   %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows
   SharedSection=1024,3072,512 Windows=On SubSystemType= Windows
   ServerDll=basesrv,1 ServerDll=winsrv:UserServerDllInitialization,3
   ServerDll= winsrv:ConServerDllInitialization,2 ProfileControl=Off
   MaxRequestThreads=16
   ```

   The desktop heap memory is defined under:

   ```
   SharedSection=1024,3072,512
   ```

5. Increase the desktop heap memory.

   The third number ("512" in the example above) is a non-interactive desktop heap. Because the non-interactive desktop heap is mapped into the address space of each and every process, this value should not be set to an arbitrarily high value, but should only be increased sufficiently to allow all the desired applications (such as Excel) to run.

"Unable to Run Macro" dialog in Excel

Reason:

Excel security settings are set too high, disallowing the client macros from running without direct user approval.

Solution:

1. In Excel, select Tools > Macro > Security.
2. Click the Security Level tab, and then click Low.

Dialog Sniffer does not log data in %SOAM_HOME%\logs\Sniffer and does not recognize FATAL_PATTERNS and NON_FATAL_PATTERNS

Reason:

Environment variable "SOAM_HOME" is not set to the "DialogSnifferService" process so the sniffer crashes when it creates a log folder or file. (The sniffer calls the ACE method and without "SOAM_HOME", it cannot load ACE.dll.) This behavior has been observed on all Windows platforms.

Solution:

1. Reboot your machine.

Client hangs when running Excel 2003 on 64-bit compute host

Reason:

Symphony Connector on compute host fails to start Excel process. This behavior has been observed only on Windows 2003 64-bit machines with Microsoft Office 2003 installed while running Symphony connector in 32-bit mode.

Solution:

1. Install the latest updates for Microsoft Office 2003 on all compute hosts in the cluster.

"This workbook has lost its VBA project ..." message in Excel 2007

Reason:

This error happens if "Visual Basic for Application" was not installed during the installation of MS Office 2007.

Solution:

1. Update your installation of MS Office:
   a) Close Excel if it is open.
   b) In the Control Panel, choose Add or Remove Programs (or Programs and Features in Windows Vista).
   c) Select Microsoft Office in the list and click Change.
   d) Choose Add or Remove Features.
   e) Select Visual Basic for Applications in the Office Shared Features list and set it to Run from My Computer. Click Continue.
   f) After the installation is finished, you will be able to use the Excel file with its VBA project when you start Excel.

Application profile

This section provides reference information for the application profile that is specific to the ConnectorForMsExcel application.

START_SNIFFER

Set this element to "true" to start Dialog Sniffer by the task if it has not yet been started. Dialog Sniffer detects dialog boxes that appear during the execution of a task, and writes the dialog box text to a log file.

Where used

Service > osTypes > osType > env

Required/Optional

Optional

Valid values

true | false

Default value

false

FATAL_PATTERNS

Only used when START_SNIFFER is set to true.

String indicating a pattern or list of patterns to search for within the log files under directory `%SOAM_HOME %\logs\sniffer`.

Separate different strings with double semi-colons (;;).

The ConnectorForExcel service searches the log file for this pattern. If the log file contains one of the specified patterns, the service terminates the Excel process started by this task and throws a fatal exception. Symphony will not attempt to rerun this task.

If no patterns are specified, the service will not search the log file for patterns.

Where used

Service > osTypes > osType > env

Required/Optional

Optional

Valid values

string

NON_FATAL_PATTERNS

Only used when START_SNIFFER is set to true.

String indicating a pattern or list of patterns to search for within the log files under directory `%SOAM_HOME%\logs\sniffer`.

Separate different strings with double semi-colons (;;).

The ConnectorForExcel service searches the log file for this pattern. If the log file contains one of the specified patterns, the service terminates the Excel process started by this task and throws a failure exception. Symphony will attempt to rerun this task up to the taskRetryLimit indicated in the session type.

If no patterns are specified, the service will not search the log file for patterns.

Where used

Service > osTypes > osType > env

Required/Optional

Optional

Valid values

string

DISMISS_DLG_WITH_PATTERNS

Only used when START_SNIFFER is set to true.

String indicating a pattern or list of patterns to search for within the log files under directory `%SOAM_HOME%\logs\sniffer`.

Separate different strings with double semi-colons (;;).

The ConnectorForExcel service searches the log file for this pattern. If the log file contains one of the specified patterns, Symphony dismisses any dialog boxes that contain text with the specified patterns. Dismissed dialog boxes are registered in the log files under directory `%SOAM_HOME%\logs\sniffer`.

If no patterns are specified, no dialogs are dismissed.

Where used

Service > osTypes > osType > env

Required/Optional

Optional

Valid values

string

FATAL_TIMEOUT

Task timeout in seconds for Fatal exceptions.

The ConnectorForExcel service throws a fatal exception when it detects that the execution time of the task exceeds the specified timeout. Before throwing this fatal exception, the service terminates the Excel process started by this task. Symphony will not retry the terminated task.

If a timeout is not specified, or if it is equal to 0, task execution time is considered unlimited.

Where used

Service > osTypes > osType > env

Required/Optional

Optional

Default value

30 seconds

NON_FATAL_TIMEOUT

Task timeout in seconds for Failure exceptions.

The ConnectorForExcel service throws a failure exception when it detects that the execution time of the task exceeds the specified timeout. Symphony will attempt to rerun this task up to the taskRetryLimit indicated in the session type.

If a timeout is not specified, or if it is equal to 0, task execution time is considered unlimited.

Where used

Service > osTypes > osType > env

Required/Optional

Optional

Default value

40 seconds

APP_DEPLOY_DIR

Do not use. Reserved for system use.

Service data flow

This section describes the data flow through the ConnectorForExcel service. Understanding these concepts will help you design your spreadsheets for optimal performance.

- In `onCreateService()`, the ConnectorForExcel service:
  1. Creates a COM object instance of ConnectorForExcel.dll.
  2. ConnectorForExcel.dll starts the Excel process and waits for task inputs from the client.
- In `onInvoke()`, the ConnectorForExcel service:
  1. Creates a thread that makes requests to Excel.
  2. Executes the macro and sends results back to the client.
  3. The `onInvoke()` method's main thread executes an event loop that waits for external events to occur such as:
     - timeouts

       The main thread throws a non-fatal or fatal exception if configured timeouts in the application profile expire. If a timeout occurs, the service terminates the Excel process started by this task before throwing a fatal or non-fatal exception.
     - pattern match in Dialog Sniffer log file

       If patterns specified in the application profile are found, terminates the Excel process started by this task and throws a non-fatal or fatal exception.
     - Dialog Sniffer dismisses dialogs with patterns

       Dialog Sniffer attempts to dismiss dialogs with patterns specified in the application profile and logs a message in the log file. ConnectorForExcel service continues execution regardless of whether dialog dismissal is successful or not.
- For non-fatal exceptions, Symphony retries the failed task.
- The VBA macro formats its result message and returns the result to the ConnectorForExcel.dll.
- The ConnectorForExcel service sends the result back to the client.
- In `onDestroyService()`, the ConnectorForExcel service cleans up and shuts down the Excel process.

Index