SAP Business One DI Event Service 2.0.1 Article

Applies To:

SAP Business One / SAP Business One SDK. For more information, visit the Business One homepage. .

Summary

This article describes how to define and implement a DI Event Service that runs on top of the existing SAP Business One SDK interfaces. This service will provide notification of events related to SAP Business One DI API objects through a listener-based interface.

Together with this article you can download an implementation sample. The development environment chosen is .NET, for both C# and VB.NET programming languages.

Author: SAP B1 Solution Architects

Company: SAP

Created on: 28 February 2011

Table of Contents

Introduction

Actually the SDK DI API does not offer a way to be notified when a DI API object has been added/updated/removed… in the database. The only accepted possibility to implement this kind of notification by partners is to place some code into the SBO_SP_TransactionNotification stored procedure; usually this code calls a dll that uses DI API methods to perform some callback actions (please refer to the SDN document "Using the SBO_SP_TransactionNotification Stored Procedure" which includes a code sample).

This approach has the main drawback that several add-ons need to add code into the SBO_SP_TransactionNotification stored procedure and this can cause confusion and difficulty when add-ons need to interwork. To avoid this issue and propose a more usable interface we have developed the SAP Business One DI Event Service. This service offers an easy to use high-level interface integrated with the SDK.

In this article we propose then a listener-based interface for data event notification together with a sample reference implementation. This code is delivered as a sample and it is not supported; you can change it in order to improve the quality of service of the proposed implementation according to the needs you have in your solutions.

Interface

The objective of this service is to provide a high-level interface integrated with the SDK and allow several add-ons to share the service in charge of the events notification.

The proposed interface offers the possibility to connect to the service and to register a method for every event your add-on wants to be notified. Once you register a method for an event, this method will be called every time the corresponding transaction is performed in the Database.

Let's suppose you want to write a new line in a file every time a new Item is added in the database. To do that you only need to connect to the B1DIEventService and to register the method adding the line in the file as a listener for the Add Items transaction. After registration, your method will be called automatically every time a new Item is added.

```
// Connect to the B1DIEventsService
B1DIEventsService diEventsService = new B1DIEventsService(oCompany);
diEventsService.Connect(ConnectionLost_Listener);

// Add a listener for the transaction Add Items
diEventsService.AddListener(SAPbobsCOM.BoObjectTypes.oItems.ToString(),
                            B1DIEventTransactionTypes.Add.ToString(),
                            AddItems_Listener)

// Declaration of the listener method adding a line in the file
public void AddItems_Listener(B1DIEventService.B1DIEventArgs eventInfo)
{
 ...
}
```

Following subsections explain one by one the main methods offered by this interface.

B1DIEventsListener Constructor

```
public B1DIEventsService(SAPbobsCOM.Company oCompany)
```

In order to connect to the service you need to create a reference of the B1DIEventsListener object.

B1DIEvents constructor receives the current company you are connected to as a parameter; it will use this company reference to obtain the database name and the location of the server. The use of this reference also controls the user has the authorization to access DI objects.

If you need to be notified for the transactions taking place in more than one database you need to create one instance of the B1DIEventsService class per each database you need to be notified.

Connect Method

```
public void Connect(B1DIEventsConnectionLostListenerDelegate listener)
```

Connects to the B1DIEventServer located in the machine where the Company database is located.

As a parameter you must specify the method to be called if a problem arrives and the connection with the server is lost. This method must follow the model fixed by the B1DIEventsConnectionLostDelegate. Next figure shows an example:

```
// Declaration of the listener method to be called if connection lost
public void ConnectionLost_Listener()
{
 ...
}
```

AddListener Method

```
public void AddListener(
                        string objType,
                        string transType,
                        B1DIEventsListenerDelegate listener)
```

The AddListener method allows you to register the method to be called by the B1DIEventService when the corresponding event occurs.

The required parameters are:

1. A string representing the object type.
   You can obtain this string from the SBObobsCOM.BoObjectTypes enum.
   Example: `SAPbobsCOM.BoObjectTypes.oBusinessPartners.ToString()`
   Notes:
   * To register a listener for all object types you need to pass an empty string.
   * If you want to add a listener for an object type not defined by BoObjectTypes you can directly use a string enclosing the number identifying the object type.
   * If you want to add a listener for an existing UDO you can place directly the code of the UDO as object type.

2. A string representing the transaction type.
   You can obtain this string from the B1DIEventsService.B1DIEventsTransactionTypes enum.
   Example: `B1DIEventsService.B1DIEventsTransactionTypes.Add.ToString()`
   In order to register a listener for all transaction types you need to pass an empty string.

3. A pointer to a delegate method defined in the add-on code by following the model fixed by the B1DIEventsListenerDelegate.
   Example:

```
// Declaration of the listener method to be called for Items Add transaction
public void AddItems_Listener(B1DIEventsService.B1DIEventArgs eventInfo)
{
  ...
```

```
// Delegate method model
public delegate void B1DIEventListenerDelegate(B1DIEventArgs eventInfo);
```

RemoveListener Method

```
public void RemoveListener(
                           string objType,
                           string transType)
```

You can stop receiving notifications on a specific event by calling the RemoveListener method.

The parameters are:

1. A string representing the object type, see AddListener first parameter.
2. A string representing the transaction type, see AddListener second parameter.

Disconnect Method

```
public void Disconnect()
```

To stop receiving all events and disconnect from the B1DIEventService you have to call the Disconnect method.

Please do not forget to call this method before you stop your add-on, otherwise the resources allocated for the connection will not be correctly freed.

Architecture

We propose here a sample reference implementation of this service. This implementation is based on a client-server architecture where:

- A server is in charge of collecting transactions information from the database and sending notifications to the registered clients.
- Add-ons that want to receive notifications use a client dll. This client dll exposes the listener based interface described above, and interact with the server hiding all the communication details to the add-ons

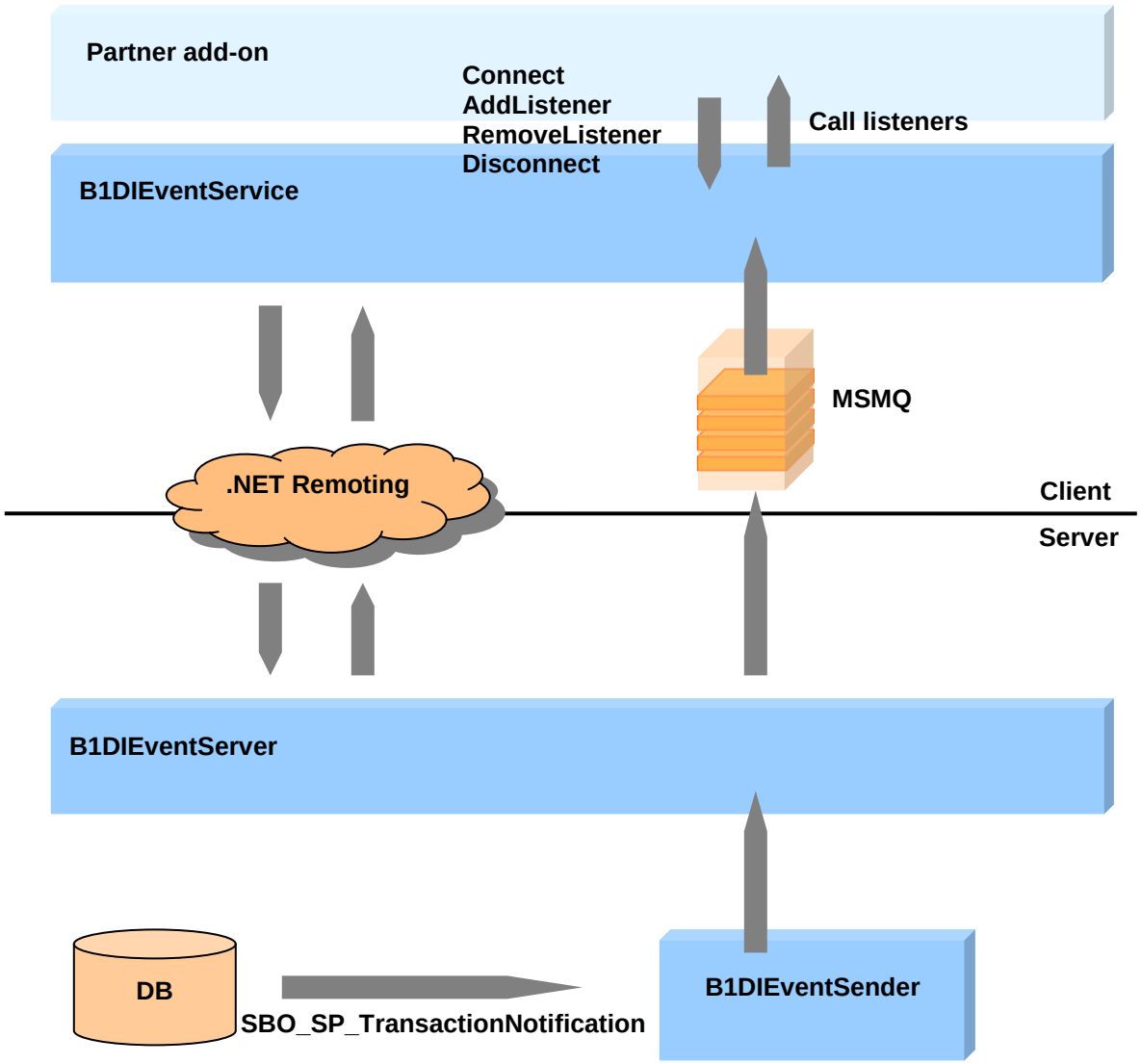
This implementation sample has been developed in C# and the interface allows using it by programming in .NET languages. Together with the service implementation we provide a couple of sample of add-ons; one in C# and another VB.NET.

The technology employed to communicate between client/server applications are:

- .NET Remoting.
- MSMQ

You can download the code sample of this implementation and change/customize the technologies used for your specific need. With this sample we want to show how easy is to create a generic server that can be used by several add-ons. You can download the setup ready to be installed and the whole source code as well.

Next figure shows the different modules composing the B1DIEventService proposed. The modules colored in blue represent the components of the B1DIEventsService, the modules in orange are external components and the light blue module represents a partner add-on using the service through the B1DIEventService interface.



Server

The server is composed of 2 modules:

- B1DIEventServer.
  Windows service application called “SAP Business One DI Event Service” that
  - Creates a .NET Remoting service.
    All clients will connect to the same .NET Remoting singleton.
  - Controls per each client the keep-alive messages.
    Each client sends periodically a keep-alive message by calling a .NET Remoting method offered by the server; the server considers a client as dead and removes all its listeners after a period of time defined in the configuration file by the element “keepalivemax” (cf. Client and Server Configuration File).
  - Per each database transaction (information received from the B1DIEventSender), posts a new message on the MSMQ of each registered client.
    - Each client has a separate MSMQ locally, when the client reads a message it is automatically removed from the queue.
    - If the network connection breaks down for a short period of time, the messages are automatically kept by MSMQ in the server side and will be sent to the client queue when the connection is re-established. No messages are then lost if the time the connection is down is inferior to the “keepalivemax” parameter defined in the configuration file (cf. Client and Server Configuration File).
    - The messages will remain in the MSMQ for a maximum number of hours defined in the configuration file by the element “messagelife” (cf. Client and Server Configuration File), after this time elapsed the messages are automatically destroyed.
- B1DIEventSender.
  Library automatically called from the SBO_SP_TransactionNotification stored procedure (cf. SBO_SP_TransactionNotification Code) inside the B1 database per each database transaction. This library will send the transaction information to the B1DIEventServer Windows Service through sockets.

B1DIEventService

The B1DIEventService is a library that both client and server must compile with (nothing to be changed by the partner). It acts as a filter taking simple commands from the client application and implementing the technical part of the connection with the B1DIEventServer.

This library offers the B1DIEventListener interface and is in charge of:

- Connect to the B1DIEventServer in the DB machine using .NET Remoting interface, a configuration file contains the registered server port number.
- Create a MSMQ locally where the server will post all events information.
- When called from the Add-On, call an equivalent function in the B1DIEventServer.
- Per each message received in the MSMQ (from the server) call the registered method in the add-on code (registering is done with AddListener method).
- Sends keep-alive messages periodically to the server.
  To avoid keeping in the server listeners that are not anymore needed (ex: client application stopped), the B1DIEventService will automatically send keep-alive messages to the server. The period between two keep-alive messages is fixed in the client and server configuration file by the "keepalive" element. The server will consider the client application as dead after a maximum period of time fixed by the element "keepalivemax" configuration file.
  To send the keep-alive messages the service calls a .NET Remoting method defined by the B1DIEventsServer.
- Tracks the number of keep-alive messages failed and alerts the client application when the number exceeds a maximum defined by "keepalivemax"/"keepalive" (both values defined in the configuration file). To alert the client the service calls the listener method given at the connection.

You can have a look to the B1DIEventsService.dll interface on annex B1DIEventService.dll, B1DIEventsService Interface.

Installation

B1DIEvents service installation is separated in two packages: one for the server and one for the client.

Installing the Server

To install B1DIEventsService server you need to run the Setup.exe inside the B1DIEventsServerSetup.

Once installed you will have a directory called B1DIEventsService/B1DIEventsServer (in the place you chose during installation) containing:

- A directory called B1DIEventsServerService.
  This directory contains a windows service executable.

Note: The service is only registered; you need to start it from the services window (Control Panel->Administrative Tools->Services).

  The server takes some seconds to stop running after stopping the service; please wait until the server is stopped to try to run it one more time (B1DIEventsServer.exe process). If between the Start/Stop service the server is not completely stopped an error message will prompt ("Cannot start the SAP Business One DI Event Service service on Local Computer.").

- A directory called SBO_SP_TransactionNotification.
  This directory contains:
  - A txt file named SBO_SP_TransactionNotification.
    You need to copy the content of this txt file into the SBO_SP_TransactionNotification already existing in your B1 database.
  - A library called B1DIEventsSender.dll.
    Once the SBO_SP_TransactionNotification modified, per each db transaction this library will be called. You don't need to do anything with this library; it is already registered and connects automatically to the server windows service.

Installing the Client

To have the B1DIEventsService client side installed you need to run the Setup.exe inside the B1DIEventsClientSetup.

Once the setup run you will have a directory called B1DIEventsService/B1DIEventsClient (in the place you chose during installation) containing:

- A directory called B1DIEventsService containing
    - The library called B1DIEventsLibrary.dll. Your add-ons must add a reference to this library in order to use the proposed interface and register your add-on to the B1DIEventsServerService.
    - A configuration xml file model enclosing information about the network connection plus settings on the traces files.
      If B1DIEventsClient is installed the registry key "`HKEY_LOCAL_MACHINE/Software/SAP/SAP Business One DIEventsService/B1DIEventsClient`" will point to the configuration file location. If you haven't installed B1DIEventsClient in a machine where your add-on will run you have two options:
        - create this registry entry during your add-on installation pointing to the right directory

          or
        - put the configuration file in the same directory as your add-on executable.
- A "Samples" directory.
  Inside it you will find two code samples, one in Vb .NET and another in C#. You can use these samples in order to guide you on how to use the B1DIEventsService.

As the events will be sent by using MSMQ technology you need to install the MSMQ Windows service in both client and server machine. The steps to follow are:

- Inside control panel select Add or Remove programs
- Click on Add/Remove Windows Components
- Select Message Queuing check box and follow the standard installation.

Configuration

In order to be able to run the B1DIEventsService you need to configure both client and server machines and to add some code into your add-on. The following sections explain in detail what needs to be done.

Server

Before being able to use the B1DIEventsService some steps must be followed:

1. Copy the code inside the text file SBO_SP_TransactionNotification.txt (Figure 2) into the SBO_SP_TransactionNotification stored procedure of the B1 database you want to work with. The SBO_SP_TransactionNotification stored procedure is created automatically when you create a new company in SAP Business One. If you need more information please have a look to the SDN CodeSample called Using the SBO_SP_TransactionNotification Stored Procedure.
2. Start the "SAP Business One DI Event Service" windows service from the Services window list (Control Panel -> Administrative Tools -> Services).
   You can change the Startup type property on the service to have it auto started when OS starts.
3. Change the configuration file if needed; cf. Client and Server Configuration File to have more information.
   Server configuration file sets the server configuration and it is located inside the B1DIEventsServerService directory. The default parameters are defined to work without any changes, if you change something in the server side please be careful and compare it to the client configuration.

A traces file is automatically created while server is running. The settings defining this file are in the configuration file. The default location of the traces file is the directory defined by the registry "`HKEY_LOCAL_MACHINE/Software/SAP/SAP Business One DIEventsService/B1DIEventsServer`".

Client

In order to connect to the B1DIEventsServerService you need to add some code in your add-on to connect to the server and register the events you want to be notified.

After B1DIEventsClient installed you have two samples resuming how to connect and register to the server, one coded in Vb .Net and another in C#. As a first step you can have a look to these samples and run them, they will show you how to use the interface.

Once you understand how the client samples are working you should be ready to add the needed code in your add-ons to manage the B1DIEventsService. Here you have the basic steps to follow:

1. Add a reference to the B1DIEventsService.dll
2. Implement the code needed to create an instance of the `B1DIEventsService` class. Please remember you need one instance of B1DIEventsService per each different database you want to receive the B1DIEvents from. The constructor of the B1DIEventsService class receives the SAPbobsCOM.Company reference the add-on is connected to as parameter; the server location will be obtained from the Company reference.
3. Add a method following the delegate model B1DIEventsConnectionLostDelegate (Figure 7: Delegate method model for the ConnectionLost event.).
4. Call the `Connect` method on the B1DIEventsService instance you have just created. You need to specify the method defined in point 3.
5. Add a method following the delegate model on the B1DIEventsService class per each event you want to be notified (Figure 5: Delegate method model for the DB transactions events.).

6. Add a listener to the B1DIEventsListener instance per each event you want to be notified by calling AddListener method. You will need to specify:
    a. The object type.
    b. The transaction type.
    c. One of the methods you have defined in point 5.
7. Add a call to the `Disconnect` method on the B1DIEventsService before the client application stops.
   It is very important to call this method before stopping your add-on, the server receives a periodically keep alive message from each client. The server considers a client as disconnected after "keepalivemax" time is elapsed. The client considers a server as dead after a time equivalent to "keepalivemax"/"keepalive" times the keep alive hasn't been received. Nevertheless during this time the server will continue sending requests to the client and will loose performance if the client is stopped without sending a disconnect message. Also connections will not be properly closed and performances can be highly affected in both client and server machines.
8. There is a configuration file for the client application, please have a look to section Client and Server Configuration File. This configuration file is located after B1DIEventsService installation on the "SAP Business One DI Event Service/ SAP Business One DI Event Client/B1DIEventsService" directory. The default parameters are already set to work without any changes, if you change something in the client side please be careful and check it is compatible with the server configuration.
   The B1DIEventsService searches for the configuration file inside the installation directory defined by the registry "`HKEY_LOCAL_MACHINE/Software/SAP/SAP Business One DIEventsService/B1DIEventsClient`". If this registry does not exist it searches in the directory where the client application runs. If you haven't installed B1DIEventsClient in a machine where your add-on will run you have two options: create this registry entry during your add-on installation with the right directory or put the configuration file in the same directory as your add-on executable.

A traces file will be automatically created while your application using the B1DIEventsService.dll is running. The settings defining this file are in the configuration file. The default location of the traces file is the directory defined by the registry "`HKEY_LOCAL_MACHINE/Software/SAP/SAP Business One DIEventsService/B1DIEventsClient`" if you have installed the B1DIEventsService in the client machine or the directory where your application runs.

Some code lines resuming the client coding steps are located in annex Client code sample.

Troubleshooting

If you have problems while installing/configuring the DIEventService you can tell us through the SAP Business One Development Forum with the prefix "B1DIEventService:". There are already several messages talking about DIEventService in the SDK forum. There is also a section in the B1 FAQs wiki page; you can contribute to this page by sharing your tips and tricks.

How to Download

From this article you can download the source code of the B1DIEventService and also the installation packages of the B1DIEventService for SAP Business One versions 2004 and 2005. For each setup version you have the setup for the client and server installation.

SAP Business One DI Event Service - Setup package for B1 2005 SP01

SAP Business One DI Event Service - Setup package for B1 2007

SAP Business One DI Event Service 2.0.1 - Setup package 32 bit for B1 8.8

SAP Business One DI Event Service 2.0.1 - Setup package 64 bit for B1 8.8

SAP Business One DI Event Service 2.0.1 - Source Code

The B1DIEventService is given as a free source code and therefore there is no support by SAP for the provided tools.

If you have any comments regarding this service please create a new message into the SAP Business One Development Forum with the prefix “B1DIEventService: ” in the title to be able to easily identify it.

Annexe

B1DIEventService.dll, B1DIEventsService Interface

```
/// Manages the connection to the B1DIEventsServer.
/// You must create one instance per Company you want to be notified
public B1DIEventsService(SAPbobsCOM.Company oCompany)
{ … }

/// Creates a network channel and connects to the B1DIEventServer
public void Connect(B1DIEventsConnectionLostDelegate listener)
{ … }

/// Adds a listener for an objectType together with a transaction type
public void AddListener(
                        string objType,
                        string transType,
                        B1DIEventsListenerDelegate listener)
{ … }

/// Removes the listener for an objectType and a transaction type
public void RemoveListener(
                           string objType,
                           string transType)
{ … }

/// Removes all listeners and disconnects from the Server
public void Disconnect()
{ … }
```

Figure 1: B1DIEventsService interface

SBO_SP_TransactionNotification Code

```
DECLARE @object int  --declare the object variable
DECLARE @hresult int --declare the hresult variable
DECLARE @db_name nvarchar(120) -- database name
Select @db_name = db_name()

EXEC @hresult = sp_OACreate 'B1DIEventsSender.Profiler', @object OUT
EXEC @hresult = sp_OAMethod @object, logDb,  NULL,
   @db_name,
   @object_type,
   @transaction_type,
   @num_of_cols_in_key,
   @list_of_key_cols_tab_del,
   @list_of_cols_val_tab_del

IF @hresult <> 0
    BEGIN
   EXEC sp_OAGetErrorInfo @object
   RETURN
    END
```

Figure 2: Code to be placed in SBO_SP_TransactionNotification.

Note: The logDb method expects a @object_type parameter of type nvarchar(20). Check that your SBO_SP_TransactionNotification stored procedure has it declared as nvarchar(20), change it if it is not the case.

Client code sample

```
Connection

// Create an instance of the listener service
evtService = new B1DIEventService(oCompany)
evtService.Connect(ConnectionLost_Listener)
// Add a listener method per each group: objType + transaction Type
evtService.addListener(SAPbobsCOM.BoObjectTypes.oItems.ToString(),
                       B1DIEventTransactionTypes.Add.ToString(),
                       AddItems_Listener)
…

Disconnection

// Remove a listener
evtService.removeListener(SAPbobsCOM.BoObjectTypes.oItems.ToString(),
                          B1DIEventTransactionTypes.Add.ToString())
// Disconnect the service
evtService.disconnect()
```

Figure 3: Client Code sample, how to register a listener.

```
// AddItems Delegate implementation in the add-ons side
public void AddItems_Listener(B1DIEventService.B1DIEventArgs eventInfo)
{
```

Figure 4: Client Code sample, how to declare a listener method for the DB transactions events.

```
// Delegate method model
public delegate void B1DIEventListenerDelegate(B1DIEventArgs eventInfo);
```

Figure 5: Delegate method model for the DB transactions events.

```
// ConnectionLost Delegate implementation in the add-ons side
public void ConnectionLost_Listener()
{
```

Figure 6: Client Code sample, how to declare a listener method for the ConnectionLost event.

```
// Delegate method model
public delegate void B1DIEventsConnectionLostDelegate();
```

Figure 7: Delegate method model for the ConnectionLost event.

Client and Server Configuration File

Partners can change several parameters for the execution of the server and client applications.

The default configuration values are set to work without any changes, if you change something in the client side please be careful and check it is compatible with the server configuration.

The configurable parameters are the same for both client and server:

- Relating to the .NET Remoting configuration
    - channelport: Port number the server registers for .NET Remoting service and the client connects to.
    - keepalive: Time elapsed between two keep-alive messages sent by the client application.
    - keepalivemax: Maximum period of time the server will wait between 2 keep-alives messages from a client. The server considers a client as disconnected after this time elapsed.
      The client considers the server as dead after keepalivemax/keepalive times the call to the keep-alive method in the server fails.

- messagelife: Number of hours the event messages are kept in the MSMQ while waiting to be read. After this number of hours elapsed the event will be automatically destroyed if the client hasn't read it before (messages are automatically removed when read from the client).

- Relating to the traces file
  - dirpath: Directory where the traces file will be created (a Log directory will be automatically created inside it). When no directory specified the Logs directory is created inside the directory where the service/server is installed.
  - level: Level to trace: Verbose, Info or Warning
  - csvseparator: Traces files will be text files with csv separators in case you want to read them with excel. The partner can choose whether they want to have a "," or a ";" separator.
  - maxfilesize: Maximum size of the traces file after which a new file will be created or the actual one will be emptied.
  - maxsizeaction: After the traces file pass the maxfilesize parameter there are two possibilities: create a new file with the actual date/time and leave the old one or replace the old one with a new empty one.

```
<?xml version="1.0" ?>
<configuration>
  <remoting>
    <channelport>4334</channelport> <!-- same for client and server (default 4334) -->
    <keepalive>2000000</keepalive> <!-- milliseconds between 2 keepalive (df. 2000000)-->
    <keepalivemax>12000000</keepalivemax> <!-- max between 2 keepalive (df. 12000000)-->
    <messagelife>36</messagelife>   <!-- nb hours messages kept in msmq (df. 36h)-->
  </remoting>
  <tracer>
    <dirpath></dirpath> <!-- Complete path (default "" = configDir/Logs) -->
    <level>Warning</level> <!-- "Verbose" "Info" "Warning" (default Warning)-->
    <csvseparator>;</csvseparator> <!-- "," ";" (default ";")-->
    <maxfilesize>6000000</maxfilesize> <!-- traces file max size (df. 6M)-->
    <maxsizeaction>Replace</maxsizeaction> <!-- CreateNew, Replace (df. Replace)-->
  </tracer>
</configuration>
```

Figure 8: Client and Server Configuration file.

Related Content

For more information, visit the Business One homepage.

Copyright