

## Easycom .NET

### Introduction



# .NET

Easycom .NET is a set of .NET managed classes that allows access to all resources of the AS/400.

Easycom .NET opens the AS/400 to the Microsoft .NET environment. It is compatible with all kind of .NET environment, i.e. .NET assemblies, .NET Windows forms or console programs, services, ASP.NET applications.

Easycom .NET includes a fast read engine for efficiently calling SQL stored procedures or retrieving data. This is possible because of using a specific subsystem and program on the AS/400, EASYCOM. This read engine is compatible with LOB fields.

Easycom .NET also includes sessions pooling (timeout and limit can be set), many advanced security options, prestart jobs, etc. This product is highly scalable and offers many configuration settings both on client and server side.

Session pooling and fast read engines ensure very good performance, especially in ASP.NET environments.

Easycom .NET is based on the Microsoft .NET Framework and implements most ADO.NET interfaces (**System.Data.EasycomClient namespace**).

Easycom .NET includes additional interfaces and methods to implement AS/400 specific tasks like (**Easycom namespace**):

- all possible native operations on an AS/400 file (Easycom.EasycomFile class)
- basic operation on an opened file (Easycom.EasycomDataSet class)
- AS400 programs call (Easycom.EacXML class)

#### Two versions of Easycom .NET exist:

1) The first one is based on **.NET Framework 4.8** (and previous versions).

This version of Easycom for .NET will be usable by Visual Studio projects targeting Microsoft .NET Framework 4.8.

Target platform is Windows only.



2) The second one is based on **.NET 9**.

.NET 9 is supported with Visual Studio 2022.

Target platforms are Windows and Linux (64-bit).

## Installation

### System Requirements

#### Server

- All AS/400 models since series B
- OS/400 from version from V5R2 to V7R5
- LOB data and SQL stored procedures features
- A PC connection using a TCP/IP network

#### Client

- Any .NET compatible device using Windows or Linux 64-bit OS (for .NET 9)
- TCP/IP protocol

#### Software

- .NET Framework 4.8. or .NET 9.
- Any .NET compatible development tool

### Installing Easycom .NET on a development environment

The installation procedure for Easycom .NET is carried out from a PC, and consists of two stages:

- Installing the client module on a PC,
- Installing the server module on an AS/400.

The AS/400 module must only be installed once on the AS/400.  
During the installation, the following operations are carried out:

#### On the PC

- Creating a folder "Easycom .NET" in the "Program files" directory,
- Creating a specific folder for the examples, (under "C:\Users\Public\Documents\Easycom For .NET Samples").
- Copies the NET assembly DLL in "Easycom .NET" directory.
- Copies Easycom core library according to the target OS.

#### On the AS/400, see "Installing Easycom Server"

- Creating a library "EASYCOM",
- Creating program and other objects in the Easycom library
- Optionally creating samples files

### Activation key

To run Easycom .NET on your AS/400, you must purchase a user license.  
You must register the serial number of the Easycom .NET product on our [website](#).

## Using Easycom for .NET

### Using Easycom in a .NET development environment

To use Easycom in a .NET environment you need to reference the main assembly of Easycom For .NET.

The assembly name is '**System.Data.EasycomClient**'.

So you need to declare a reference to this assembly 'System.Data.EasycomClient'. If the assembly does not appear in the list, you can manually select the assembly DLL :

Program Files (x86)\Easycom. NET\System.Data.EasycomClient.dll

The assembly '**System.Data.EasycomClient**' contains the 2 main following namespaces:

- 1) **Easycom**: methods to implement AS/400 specific tasks
  - 2) **System.Data.EasycomClient**: contains some inherited classes that makes Easycom .NET an ADO.NET compatible assembly.
- So, for some topics that may miss from this documentation please refer to an ADO.NET documentation.

Then you should be able to instantiate some Easycom .NET classes which are for instance:

EacConnection (implements System.Data.IDbConnection)  
EacTransaction (implements System.Data.IDbTransaction)  
EacCommand (implements System.Data.IDbCommand)  
EacDataReader (implements System.Data.IDataReader)  
EacDataAdapter (implements System.Data.IDbDataAdapter)  
EacParameterCollection  
EacParameter

First you must instantiate an EacConnection Object. Then you can create the other objects linked to it. At this point you can choose between generic or specific coding (using interfaces or using EacXX classes).

For example, in C# coding:

```
EacConnection mySqlConnection = new EacConnection();  
mySqlConnection.Open();  
  
EacCommand mySqlCommand = mySqlConnection.CreateCommand();  
mySqlCommand.CommandText = "select * from s_customer order by Cust_id";  
  
EacDataReader myReader = (EacDataReader) mySqlCommand.ExecuteReader();  
[...] use myReader...
```

Example, in VB coding:

```
Dim mySqlConnection As EacConnection  
mySqlConnection = New EacConnection
```

```
mySqlConnection.Open()

Dim mySqlCommand As EacCommand
mySqlCommand = mySqlConnection.CreateCommand()
mySqlCommand.CommandText = "select * from s_customer order by Cust_id"

Dim myReader As EacDataReader
myReader = mySqlCommand.ExecuteReader()
[...] use myReader...
```

This is even possible to do it at runtime (because the only difference is when creating the IDbConnection-derived object).

### Connection pooling

Easycom for .NET allows you to simply manage pooled connections. It is recommended to use pooled connection when working using a n-tiers architecture such as ASP.NET.

In fact, with Easycom .NET the connections are pooled by default. This means that a connection will remain "alive" even when the connection object is no longer user or when the "Close()" method is called.

The connection link remains active between the Windows process and the AS/400 job.

When a new connection is requested by the same process (on same or different thread), a pooled connection is used if there is one available. This only works if the connection has the same properties (User ID, Password, Server Name, CodePage File, SqlNaming, Init Libl, JobName, EacUnlock), otherwise a new connection is created if the PoolLimit parameter allows that.

Indeed, it is possible to limit the total number of jobs that are pooled using the **PoolLimit** parameter. **This parameter is only usable on Windows platforms.**

It is possible to determine a maximum time to wait to enter in a pool (in case a pool limit is defined), using the PoolTimeout parameter. If the timeout is reached a specific exception is thrown (using the EasycomException class).

### Deployment of an application/assembly based on Easycom for .NET

The .NET Framework is mandatory on the target system.

The deployment of a program based on Easycom .NET is quite simple.

You need to deploy the .NET assembly file/executable , and some DLL files:

- **For 32-bit Windows systems**, the files to deploy are:
  - C:\Program Files (x86)\Easycom .NET\System.Data.EasycomClient.dll : the main assembly
  - C:\Program Files (x86)\Easycom .NET\Win32\Easycom\_core.dll : the x86 core file
- **For 64-bit Windows systems**, the files to deploy are:
  - C:\Program Files (x86)\Easycom .NET\System.Data.EasycomClient.dll : the main assembly
  - C:\Program Files (x86)\Easycom .NET\x64\Easycom\_core.dll : the x64 core file
- **For 64-bit Linux systems and .NET 9**, the files to deploy are:

- C:\Program Files (x86)\Easycom .NET\System.Data.EasycomClient.dll : the main assembly
- C:\Program Files (x86)\Easycom .NET\linux64\Easycom\_core.dll : the 64-bit core library

## System.Data.EasycomClient namespace

### Introduction

The “**System.Data.EasycomClient**” namespace implements most ADO.NET interfaces.

ADO.NET is an integral part of the .NET Framework and contains a set of classes that expose data access services for .NET Framework programmers.

Standard documentation of ADO .NET can be found here:

[https://msdn.microsoft.com/en-us/library/e80y5yhx\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/e80y5yhx(v=vs.110).aspx)

In this chapter the main classes contained into the “**System.Data.EasycomClient**” namespace are described:

- EacConnection
- EacTransaction
- EacCommand
- EacDataReader
- EacParameter

### EacConnection

#### EacConnection Class

This object is derived from System.Data.IDbConnection and EasycomConnection interfaces. EasycomConnection is a class defined in the Easycom namespace and is used for Native access functions.

This class represents a physical connection to the AS/400. Because this connection can be "pooled", when you close the object (using .Dispose() or .Close() method) the physical connection may remain.

#### EacConnection properties

##### EacConnection.Database

```
public string Database [ get]
```

This property contains the IP address or name of the AS/400 to connect to. This is the same as the 'Server' entry in the EacConnection.ConnectionString property.

If this property contains a name instead of the IP address, the name will be solved using the standard TCP/IP host resolving process (using host table, DNS, ...)

If no value is specified in this parameter, the default value is read from the easycom.ini file. And if no value is specified in the easycom.ini file a dialog box will prompt the user to enter a valid IP address or name. If the assembly is used in an NT service mode no dialog box will

be shown, and an exception will come up showing the corresponding TCP/IP error (Easycom.Core.EasycomException).

### EacConnection.ConnectionString

```
public string ConnectionString [ get, set ]
```

This property is designed to set-up connection-related properties before calling the IDbConnection.Connect() method.

The available key words are:

- **Server:** The name or the IP address of the AS/400
- **User ID:** The user profile. In interactive environment, if this parameter is omitted an Easycom dialog box will be shown. If this parameter is omitted in a service like program (for example in ASP.NET environment), an exception will be thrown during the connection.
- **Password:** The password that belongs the User ID.
- **Init Libl:** Some libraries to add to the default library list (that belong to the user ID). This is a semicolon-separated list, such as "MYLIB1;MYLIB2".
- **JobName:** The job name to use. By default this is the name of the PC.
- **EacUnLock:** A special password that can be validated by a user-defined AS/400 program. This avoids to non-authorized developers to use Easycom. See 'Advanced security features' in the server documentation.
- **CodepageFile:** designates an ASCII-EBCDIC translation file. This is mostly useful for DBCS countries like China, Japan, Korea.
- **ShortFieldNames:** tells Easycom to use only short forms for field names. This will ignore aliases of columns. Valid values are 'True' and 'False'. Default value is False.
- **SqlNaming:** Choose between '.' and '/' to separate files and libraries. When using '.' the library list is not available (and current library is the name of the user). Possible values are '/' and '.'. Default is '/'.
- **Pooled:** Choose if pooling or not this connection object. See 'Pooling connection' topic for more details. Possible values are 'True' and 'False'. Default value is True.
- **PoolLimit:** Determines the limit of using pooled connection. When the limit is reached the thread will wait until a pooled connection is available or until the timeout is reached (see PoolTimeout parameter and 'Pooling connections' topic for more details). Non-pooled connections are not counted with this parameter. This value is an integer representing the maximum number of connections being pooled. Default is 0 (unlimited). **Windows only.**
- **PoolTimeout:** Represents the maximum number of seconds to wait if the pool limit is reached. Default value is 0 (unlimited). **Windows only.**
- **Shared:** Choose if sharing the connection for the same thread. Possible values are 'True' and 'False'. Default value is False.
- **CommandTimeout:** The CommandTimeout is configurable for a given connection. If the timeout is reached for a command, the following Easycom exception is raised: *"Easycom error, Generic error: Timeout has been reached."*

Example with Timeout = 120s:

```
Connection.ConnectionString = "Server=" + "power8" + ";User Id=" + "aura" +  
";Password=" + "aura" + ";CommandTimeout=120"
```



The **ConnectionString** value is a semicolon separated list of parameters. The value is read/write when the connection is closed and read-only when the connection is opened.

The value of the **ConnectionString** may change to fit a standard syntax and if some parameters are not recognized (i.e., you may not retrieve the value you have put). When developing you can read back the **ConnectionString** value to check if all parameters are understood as expected.

Example of usage (C#) :

```
EacConnection cnx;  
cnx = new EacConnection();  
cnx.ConnectionString = "User Id=qpgmr;Password=qpgmr;Init Libl=  
cnx.Open();
```

Example of usage (VB) :

```
Dim cnx As EacConnection  
cnx = New EacConnection  
cnx.ConnectionString = "User Id=qpgmr;Password=qpgmr;Init Libl=  
cnx.Open()
```

### *EacConnection methods*

#### **EacConnection.EacConnection(System.String sConnString)**

Constructs an EacConnection object with a specific connection string.

#### **EacConnection.EacConnection()**

Constructs an EacConnection object, with no connection string or default server.

#### **EacConnection.Open()**

Opens the connection, according to the EacConnection.ConnectionString parameters.  
If there is any problem an exception is thrown (Easycom.Core.EasycomException if the problem is Easycom-released, or any standard exception).

#### **EacConnection.Close()**

Closes the connection. If the connection was created with Pooling enabled, the connection remains physically active.

All objects that may stay linked to the EacConnection object become unconnected.

#### **EacConnection.Dispose()**

Frees all memory used by the EacConnection object. Also closes the connection.

#### **EacConnection.BeginTransaction**

```
public EacTransaction BeginTransaction ( )  
public EacTransaction BeginTransaction ( System.Data.IsolationLevel level  
)
```





Allocates a new transaction object, with *level* isolation state if specified (default is ReadCommitted)

The transaction is managed by the returned EacTransaction object.

### EacConnection.CreateCommand()

```
public EacCommand CreateCommand ( )
```

This creates an EacCommand object that is automatically linked to the EacConnection object. This is an alternative to the EacCommand constructor that takes an EacConnection object as a parameter (useful when doing generic programming).

### EacConnection.RemoteCommand(System.String)

```
public void RemoteCommand ( System.String strCommand )
```

This method is designed to run an AS/400 command. If the command fails, an Easycom.Core.EasycomException is thrown.

Example (VB):

```
Try
    mycnx.RemoteCommand("ADDLIB MYLIB")
Catch except As Exception
    MessageBox.Show("Error while executing command : " +
        except.Message, Me.Text)
End Try
```

Example (C#):

```
Try
{
    mycnx.RemoteCommand("ADDLIB MYLIB");
}
Catch (EasycomException e)
{
    MessageBox.Show("Error while executing command : " + e.Message);
}
```

See also: EasycomConnection.RemoteCommand

### EacConnection.RemoteRtvCommand(System.String)

```
public void RemoteRtvCommand ( System.String strRtvCommand )
```

This method is designed to call "retrieve" kind of commands. This is useful to call simple programs and retrieve some data from them. For more complex program calls, you should use stored procedures (Easycom or SQL procedures).

In this example we use a system command (RTVJOBA) to retrieve some AS/400 specific options.

Example (VB):

```
mycnx.RemoteRtvCommand("RTVJOBA USER(&USR) USRLIBL(&USRL) ")

If mycnx.RemoteRtvCommandGetValue("RC") = "0" Then
    l_user.Text = mycnx.RemoteRtvCommandGetValue("USR")
    l_job_usrlibl.Text = mycnx.RemoteRtvCommandGetValue("USRL")
Else
    MessageBox.Show("Error returned by Rtv Command: " +
        mycnx.RemoteRtvCommandGetValue("RC"), Me.Text)
End If
```

Example (C#):

```
mycnx.RemoteRtvCommand("RTVJOBA USER(&USR) USRLIBL(&USRL) ");

If (mycnx.RemoteRtvCommandGetValue("RC") == "0")
{
    l_user.Text = mycnx.RemoteRtvCommandGetValue("USR");
    l_job_usrlibl.Text = mycnx.RemoteRtvCommandGetValue("USRL");
}
Else
{
    MessageBox.Show("Error returned by Rtv Command: " +
        mycnx.RemoteRtvCommandGetValue("RC"));
}
```

## EacConnection.RemoteRtvCommandGetValue(System.String)

This function retrieves the result variables that were used in the RemoteRtvCommand call.

See EacConnection.RemoteRtvCommand(System.String)

## EacTansaction

### EacTransaction Class

This object is derived from System.Data.IDbTransaction and other interfaces.

This class represents an active transaction on the AS/400. The transaction starts when the object is created.

### EacTransaction properties

#### EacTransaction.Connection

```
public IDbConnection Connection [ get]
```

This property is a link to the connection to which the transaction belongs. This property is Read-Only.



### EacTransaction.IsolationLevel

```
public System.Data.IsolationLevel IsolationLevel [ get]
```

This property represents the transaction isolation level. This property is Read-Only.

There are several isolation level options; the following list shows the possible values for IsolationLevel and the corresponding AS/400 isolation levels:

- Chaos \*NONE
- ReadCommitted \*CS
- ReadUncommitted \*CHG
- RepeatableRead \*ALL
- Serializable \*ALL (\*RR not supported)
- Unspecified \*CS

To know all details about differences between \*CS, \*CHG and \*ALL, you can consult the online help of the LCKLVL parameter of the STRCMTCTL command on the AS/400.

In summary :

- \*NONE means that the transaction has no effect
- \*CS means that all records changed inside a transaction remain locked. The records that are read during the transaction are locked only during the fetch (only the current record can be locked). The record changes (uncommitted) cannot be seen from other jobs until the transaction is committed.
- \*CHG means that records changed inside a transaction remain locked. The records that are read for update are locked. If the record is not modified (or removed or added) the record is unlocked. The record changes (uncommitted) can be seen from other job when being inside the transaction.
- \*ALL means that all fetched records during the transaction are locked, even if not modified. The record changes (uncommitted) cannot be seen from other jobs until the transaction is committed.
- \*RR means that all objects implied by the transaction are completely locked. The record changes (uncommitted) cannot be seen from other jobs until the transaction is committed.

### EacTransaction methods

#### EacTransaction.EacTransaction

```
public EacTransaction (EacConnection connection ,  
System.Data.IsolationLevel iso )  
public EacTransaction (EacConnection connection )
```

Constructs an EacTransaction object, using a valid EacConnection object and optionally an isolation level.

The object can also be created by calling EacConnection.BeginTrans method.

The default isolation level is ReadCommitted (see EacTransaction.IsolationLevel).

The transaction is started as soon as the object is created. The transaction is ended with Commit() or Rollback() call. The transaction object has no use after the commit or rollback call.

## EacTransaction.Commit

```
public virtual new void Commit ( )
```

Commits the transaction. The commit may fail (for example because integrity checks failed), and in this case the transaction is rolled back, and an exception is thrown.

Full details of the exception are available using the EasycomException object.

Note: only '**journaled**' files are affected by a transaction.

To know if a file is 'journaled' you can use the DSPFD command. To start journaling for a file use STRJRNP command.

## EacTransaction.Rollback

```
public virtual new void Rollback ( )
```

Roll backs the transaction.

Note: only '**journaled**' files are affected by a transaction.

To know if a file is 'journaled' you can use the DSPFD command. To start journaling for a file use STRJRNP command.

## EacCommand

### *EacCommand Class*

This object is derived from System.Data.IDbCommand and other interfaces.

This is used to prepare a request to the AS/400. It can be one of the following request type:

- an SQL query to select records
- an SQL query without result (like DELETE, INSERT, ...)
- a simple AS/400 file
- an SQL Stored Procedure
- a native program, represented by an 'Easycom' Stored Procedure.

Some types of "EacCommand" can be used with input and/or output parameters.

### *EacCommand properties*

#### EacCommand.CommandText

```
public string CommandText [ get, set ]
```

This property contains the command representation as text. The text syntax must correspond to the EacCommand.CommandType property value.

(see EacCommand.CommandType)

#### EacCommand.CommandType

```
public System.Data.CommandType CommandType [ get, set ]
```

This property designates the type of request the command object is representing.

The possible values are:

- **Text** (default): Any SQL query. Can be "SELECT " or another SQL query. The EacCommand.CommandText must contain the SQL text of the query.  
The syntax must be AS/400 SQL compliant. The parameters must be specified with the ':NAME' syntax. For example:

```
cmd.CommandText="SELECT * FROM S_CUSTOMERS WHERE CUST_ID = :PCUSTID"  
(so the parameter name is PCUSTID)
```

- **StoredProcedure** : an SQL or an Easycom stored procedure. The EacCommand.CommandText must contain the procedure name only. It can be:
  - \*PGM/PROC if the procedure is an Easycom procedure (see ['Easycom Stored Procedure configuration'](#) to define an Easycom Stored Procedure).
  - LIB/PROC or PROC if the procedure is an SQL stored procedure.
- **TableDirect** : a physical, or logical file. The CommandText value must contain the name of the file, like LIB/FILE or FILE or LIB/FILE(MEMBER) or FILE(MEMBER).

## EacCommand.Parameters

```
public.EacParameterCollection Parameters [ get]
```

This property is used to define or consult the parameters collection.

Note: When using SQL stored procedures, it is recommended to define the parameters before preparing the command object. In this case the definition of the parameters can be partial or with different data type, but must correspond to the right number of parameters, and the specified data type and sizes must be compatible with the actual types and sizes.

When using Easycom stored procedures, there is no need to define the parameters before calling Prepare() method.

After the EacCommand.Prepare method call, the Parameters collection is updated.

### Example (VB.NET):

```
mycmd.CommandType = CommandType.StoredProcedure  
mycmd.CommandText = "S_ORD_SUM"  
With mycmd.Parameters.Add("CUSTOMER_NO", DbType.Int32, 4)  
.Direction = ParameterDirection.Input  
.Scale = 0  
End With  
With mycmd.Parameters.Add("total_orders", DbType.Decimal, 8)  
.Direction = ParameterDirection.Output  
.Scale = 2  
End With  
With mycmd.Parameters.Add("num_orders", DbType.Int32, 4)  
.Direction = ParameterDirection.Output  
.Scale = 0  
End With  
  
mycmd.Prepare()
```

Example (C#):

```
EacCommand mycmd = new EacCommand();
mycmd.CommandType = CommandType.StoredProcedure;
mycmd.CommandText = "S_ORD_SUM";
EacParameter param1 = mycmd.Parameters.Add("CUSTOMER_NO",
DbType.Int32, 4);
param1.Direction = ParameterDirection.Input;
param1.Scale = 0;
param1.Value = 1001;

EacParameter param2 = mycmd.Parameters.Add("total_orders",
DbType.Decimal, 8);
param2.Direction = ParameterDirection.Output;
param2.Scale = 0;

EacParameter param3 = mycmd.Parameters.Add("num_orders", DbType.Int32,
4);
param3.Direction = ParameterDirection.Output;
param3.Scale = 0;
mycmd.Prepare();

// Another way to pass value to parameter
mycmd.Parameters["CUSTOMER_NO"].Value = 1001
```

Note:

When you add a parameter, it is also possible to affect a value for the parameter. Therefore, it is not necessary to specify the type, because the “value” is used.

For a String parameter:

```
mycmd.Parameters.Add(new EacParameter("COMPANY", "AURA"));
```

For an integer:

```
Int32 cust_id = 777;
mycmd.Parameters.Add("CUSTOMER_NO", cust_id);
```

## EacCommand.Transaction

```
Public IDbTransaction Transaction [ get, set ]
```

This property is used to define or consult the parameters collection. Setting or getting this property does not change the transaction state. The EacTransaction object must be linked to the EacConnection object that is in the EacCommand.Connection property.

## EacCommand methods

### EacCommand.EacCommand

```
public EacCommand ( )
public EacCommand ( System.String cmdText )
public EacCommand ( EacConnection connection )
public EacCommand ( System.String cmdText, EacConnection connection )
```



```
public EacCommand ( System.String cmdText, EacConnection connection,  
EacTransaction txn )
```

Constructs an EacCommand Object. The object has multiple constructors.

However, the object must have a valid EacConnection and a valid CommandText defined to work properly.

The object can also be created using EacConnection.CreateCommand(). In this case the Connection property is automatically set.

### EacCommand.ExecuteNonQuery()

```
public System.Int32 ExecuteNonQuery ( )
```

This method executes immediately the command. It prepares it and passes parameters if needed, executes the query, and updates the output parameters.

With this method, Easycom will not try to retrieve data.

Typical use of this method is calling simple programs or running "UPDATE" like SQL queries.

The function returns the number of affected rows or 0 if no rows have been modified.

### EacCommand.ExecuteReader()

```
public System.Data.IDataReader ExecuteReader ( )
```

This method immediately executes the command. It prepares it and passes parameters if needed, executes the query, and updates the output parameters.

It returns an EacDataReader object that allows fetching data and selecting next resultsets if any.

If there is no resultset at all, the function returns null.

Example (C#):

```
EacConnection cnx;  
cnx = new EacConnection();  
EacCommand cmd = cnx.CreateCommand();  
cmd.CommandText = "SELECT * FROM Table";  
EacDataReader data = (EacDataReader)cmd.ExecuteReader();
```

### EacCommand.Prepare()

```
public void Prepare ( )
```



Explicitly prepares the command. This is useful to validate the "Parameters" collection. Prepare is designed to be called only once for a given EacCommand object.

One call of Prepare() method can be used before multiple calls of ExecuteReader() or ExecuteNonQuery() methods.

Example (C#):

```
String data[] = new string[2];
data[0] = "param";
data[1] = "param2";
EacCommand cmd = this.oCnx.CreateCommand();
cmd.CommandText = "SELECT * FROM Table";
cmd.Prepare();

for (int i = 0; i < data.Length; i++)
{
    EacParamter param = new EacParamter ("param" +
    i.ToString(), data[i]);
    cmd.Parameters[param.ParameterName] = param;
}

cmd.ExecuteNonQuery();
```

## EacCommand.Dispose()

```
public void Dispose ( )
```

Explicitly un-prepare the object, and free it. The object should not be used after this call.

### *Example 1: call a program*

**Into this example, the EASYCOMXMP/RPCSAMPLE program is called.**

This example is provided with [demo samples](#).

Parameters are filled by using value filled within textbox component.

OP1 is an input parameter.

STR1 is an input parameter.

OP2 is an input/output parameter. After call, OP2 = OP2 + OP1.

STR2 is an input/output parameter. After call, STR2= STR1.

OP3 is an output parameter. After call, OP3 = OP1 \* OP2.

```
Using easyConn As System.Data.EasycomClient.EacConnection = New
System.Data.EasycomClient.EacConnection()
easyConn.ConnectionString = "Server=power8;User
Id=aura;Password=aura;Init Libl=easycomxmp"
easyConn.Open()
```





```
Using easyComm As System.Data.EasycomClient.EacCommand = New
System.Data.EasycomClient.EacCommand()

With easyComm

    .Connection = easyConn
    .CommandType = CommandType.StoredProcedure
    .CommandText = "**PGM/RPCSAMPLE"

    With .Parameters.Add("OP1", DbType.Double, 1)
        .Direction = ParameterDirection.Input
        .Value = OP1.Text
    End With
    With .Parameters.Add("STR1", DbType.StringFixedLength, 1)
        .Direction = ParameterDirection.Input
        .Value = STR1.Text
    End With
    With .Parameters.Add("OP2", DbType.Double, 1)
        .Direction = ParameterDirection.InputOutput
        .Value = OP2.Text
    End With
    With .Parameters.Add("STR2", DbType.StringFixedLength, 1)
        .Direction = ParameterDirection.InputOutput
        .Value = STR2.Text
    End With
    With .Parameters.Add("OP3", DbType.Double, 1)
        .Direction = ParameterDirection.Output
    End With

End With

easyComm.Prepare()
easyComm.ExecuteNonQuery()

OP1.Text = easyComm.Parameters.Item(0).Value.ToString()
STR1.Text = easyComm.Parameters.Item(1).Value.ToString()
OP2.Text = easyComm.Parameters.Item(2).Value.ToString()
STR2.Text = easyComm.Parameters.Item(3).Value.ToString()
OP3.Text = easyComm.Parameters.Item(4).Value.ToString()

End Using
easyConn.Close()

End Using
```

### *Example 2: call a stored procedure*

**Into this example, we execute a stored procedure: CR/CUST12.**

The CR/CUST12 procedure has been created on IBMi by the following way, by using IBM I Access Client Solution:

```
CREATE PROCEDURE CR.CUST12 ( )
LANGUAGE SQL
SPECIFIC CR.CUST8
```



```
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
SET OPTION ALWBLK = *ALLREAD ,
ALWCPYDTA = *OPTIMIZE ,
COMMIT = *CHG ,
DFTRDBCOL = *NONE ,
DYNDFTCOL = *NO ,
DYNUSRPRF = *USER ,
SRTSEQ = *HEX
P1 : BEGIN ATOMIC
UPDATE EASYCOMXMP.SP_CUST SET LASTNAME= 'AURA2' WHERE FIRSTNAME= 'Ted' ;
END P1 ;
```

```
EacCommand sqlCommand = ConnectionIBM.CreateCommand();

sqlCommand.CommandText = "CR/CUST12";
sqlCommand.CommandType = CommandType.StoredProcedure;

sqlCommand.Prepare();
sqlCommand.ExecuteNonQuery();
```

### *Example 3: call a stored procedure with input parameter*

**Into this example, we execute a stored procedure: CR/CUSTPARAM3.**

The CR/CUSTPARAM3 procedure has been created on IBMi by the following way, by using IBM I Access Client Solution:

```
CREATE PROCEDURE CR.CUSTPARAM3 ( IN name CHAR(4))
LANGUAGE SQL
SPECIFIC CR.CUSTPARAM3
NOT DETERMINISTIC
MODIFIES SQL DATA
CALLED ON NULL INPUT
SET OPTION ALWBLK = *ALLREAD ,
ALWCPYDTA = *OPTIMIZE ,
COMMIT = *CHG ,
DFTRDBCOL = *NONE ,
DYNDFTCOL = *NO ,
DYNUSRPRF = *USER ,
SRTSEQ = *HEX
P1 : BEGIN ATOMIC
```



```
UPDATE EASYCOMXMP.SP_CUST SET LASTNAME=name WHERE FIRSTNAME='Louis';  
END P1 ;
```

```
EacCommand sqlCommand = ConnectionIBM.CreateCommand();  
  
sqlCommand.CommandText = "CR/CUSTPARAM3";  
sqlCommand.CommandType = CommandType.StoredProcedure;  
  
///1st method to fill input parameter  
EacParameter param1 = sqlCommand.Parameters.Add("name",  
    DbType.String, 4);  
param1.Direction = ParameterDirection.Input;  
param1.Scale = 0;  
param1.Value = "TU";  
  
// 2nd method  
string test = "TA";  
sqlCommand.Parameters.Add(new EacParameter("name", test));  
  
// 3rd method  
sqlCommand.Parameters.Add(new EacParameter("name", "TO"));  
  
sqlCommand.Prepare();  
sqlCommand.ExecuteNonQuery();
```

## EacDataReader

### EacDataReader Class

The EacDataReader implements all the functions and properties of *IDataReader*, except *Depth* and *GetGuid*.

This class is used to sequentially fetch the data returned by an *EacCommand* object. *EacDataReader.Next* method fetches the records one by one.

Data can be read using the type-dedicated functions like *GetInt32()*, or using *GetValue* or *GetValues()*.

You also can select the subsequent result sets using *EacDataReader.NextResultSet*.

Types and names can be retrieved using *GetFieldType()*, *GetName()*, *GetOrdinal()*, *GetDataTypeName()*.

Null state can be retrieved using *IsDBNull()* function.

LOB values can be retrieved using *GetData()* or *GetBytes()* functions.

### Example



A full example is provided with [demo samples](#).

Otherwise, this is another example:

```
EacCommand cmd = ConnectionIBM.CreateCommand();
cmd.CommandText = "SELECT * FROM SP_CUST";
EacDataReader dr = (EacDataReader)cmd.ExecuteReader();

//To read data
List<object[]> tab = new List<object[]>();
object[] result = null;

while (dr.Read())
{
    result = new object[9];
    dr.GetValues(result);
    tab.Add(result);
}

foreach (object[] results in tab)
{
    string first = results[0].ToString();
    MessageBox.Show(first);
}
```

### *Example with EacDataTable*

Into this example, a **EacDatatable** is used to load a **EacDataReader**.

```
string resultContent = "";

using (var cmd = ConnectionIBM.CreateCommand())
{
    cmd.CommandText = "SELECT * FROM EASYCOMXMP/SP_CUST where  
                        (CUST_ID='C-17' OR CUST_ID='C-03')";
    using (EacDataReader dr = (EacDataReader)cmd.ExecuteReader())
    {

        EacDataTable dataTable = new EacDataTable();

        dataTable.Load(dr);

        foreach (DataRow row in dataTable.Rows)
        {
            for (int i = 0; i < dataTable.Columns.Count; i++)
            {
                resultContent = resultContent + " - " +  
                                row[i].ToString();
            }
            resultContent = resultContent + Environment.NewLine;
        }
    }
}
```



```
    }  
  
    MessageBox.Show(resultContent.ToString());  
    dr.Close();  
}  
cmd.Dispose();  
}
```

## EacParameter

### *EacParameter Class*

This class represents a parameter for an SQL Query, an SQL stored procedure or an Easycom Stored procedure.

It implements IDataParameter but also contains an AS/400 specific property: EacType.

The properties of this class reflect data, datatype, size and scale of a parameter. It also contains "Source" information in

### EacParameter.EacType

```
public Easycom.Types.EasycomType EacType [ get, set ]
```

Defines the type of a parameter. This type must be compatible with the actual type of the procedure parameter.

When this property is changed, the EacParameter.DbType is automatically adjusted to the corresponding System.Type.DbType enumeration value.

The possible types are :

EasycomType	Comments	Possible Size	DbType
-------------	----------	---------------	--------

Binary	Binary Data	32000	Binary
Blob	Binary large object	Any	Binary
Char	Ansi string	32000	AnsiStringFixedLength
Clob	Character large object	Any	String
Date			Date
Decimal			Decimal
Double		8	Double
Float	8 bytes floating point	4	Float
Int64	4 bytes floating point	8	Int64
Int	8 bytes integer	4	Int32
Short	4 bytes integer	2	Int16
String	2 bytes integer	32000	StringFixedLength
Time	Unicode string		Time
TimeStamp			DateTime
VarChar		32000	AnsiString
VarString	Ansi variable string	32000	String
	Unicode variable string		

### Example

Different parameters are used into the INSERT SQL request:

First syntax:

```
EacCommand cmd = ConnectionIBM.CreateCommand();

cmd.CommandText = "INSERT INTO SP_CUST (FIRSTNAME, LASTNAME, CITY)
VALUES(:param0, :param1, :param2)";

cmd.Prepare();

EacParameter param0 = new EacParameter("param0", "rob");
cmd.Parameters["param0"] = param0;

EacParameter param1 = new EacParameter("param1", "ced");
cmd.Parameters["param1"] = param1;

EacParameter param2 = new EacParameter("param2", "sophia");
cmd.Parameters["param2"] = param2;

cmd.ExecuteNonQuery();
```

Second syntax: the Prepare is put after having added the parameters

```
EacCommand cmd = ConnectionIBM.CreateCommand();

cmd.CommandText = "INSERT INTO SP_CUST (FIRSTNAME, LASTNAME, CITY)
VALUES(:param0, :param1, :param2)";
```

```
EacParameter param0 = cmd.Parameters.Add("param0", DbType.AnsiString, 10);  
param0.Direction = ParameterDirection.Input;  
param0.Value = "ced";  
  
EacParameter param1 = cmd.Parameters.Add("param1", DbType.AnsiString, 10);  
param1.Direction = ParameterDirection.Input;  
param1.Value = "ced";  
  
EacParameter param2 = cmd.Parameters.Add("param2", DbType.AnsiString, 15);  
param2.Direction = ParameterDirection.Input;  
param2.Value = "16";  
  
cmd.Prepare();  
  
cmd.ExecuteNonQuery();
```

## Easycom namespace

### Introduction

In addition to the “System.Data.EasycomClient” namespace that implement ADO.NET methods, some native classes are provided within the “Easycom” namespace.

Those native classes are part of the System.Data.EasycomClient assembly, but in the Easycom namespace.

In this chapter the main classes contained into the “Easycom” namespace are described:

- EasycomType
- EasycomConnection
- EasycomDataSet
- EasycomFile
- EacXML
- EasycomException

## EasycomType

### EasycomType Class

#### Namespace Easycom

#### Syntax

```
public class EasycomType
```

#### Description

The EasycomType class represents different possible types compatible with Easycom.

See also [EacParameter.EacType](#) to C# equivalency.

## EasycomConnection

### *EasycomConnection Class*

Namespace Easycom

#### Syntax

```
public class EasycomConnection : System.Object
```

#### Description

The EasycomConnection class represents a physical connection to the AS/400. Because this connection can be "pooled", when you close the object (using .Dispose() or .Close() method) the physical connection may remain open.

"System.Data.EasycomClient.EacConnection" is derived from this class. So, all properties and methods of EasycomConnection are accessible from an EacConnection (ADO .NET Driver) object.

A typical use of EasycomConnection object consists of filling the "ConnectionString" property", and calling the "Open" method.

#### Example of use (C#)

```
EasycomConnection my_cnx;  
EasycomFile my_file;  
string result;  
  
my_cnx = new EasycomConnection();  
my_cnx.ConnectionString = "Server=194.206.160.111;User. "  
my_cnx.Open();  
my_file = my_cnx.OpenFile("S_CUSTOMER");  
if (my_file.ReadKey(2156))  
{  
    result = my_file.GetValue("COMPANY").ToString() +  
    " - " + my_file.GetValue("FIRSTNAME").ToString();  
    MessageBox.Show("Customer found : "+result);  
}  
else  
    MessageBox.Show("Customer not found!");
```

#### Example of use (VB.NET)

```
Dim my_cnx As EasycomConnection  
Dim my_file As EasycomFile  
Dim result As String  
  
my_cnx = New EasycomConnection  
my_cnx.ConnectionString = "Server=194.206.160.111;User"  
my_cnx.Open()
```





```
my_file = my_cnx.OpenFile("S_CUSTOMER")
If (my_file.ReadKey(2156)) Then
    result = my_file.GetValue("COMPANY").ToString() + "
    - " + my_file.GetValue("FIRSTNAME").ToString()
    MessageBox.Show("Customer found : " + result)
Else
    MessageBox.Show("Customer not found!")
End If
```

### *EasycomConnection Constructor*

Creates a new instance of an EasycomConnection object, which represents an open connection to a specified AS/400.

**Class** EasycomConnection

#### **Syntax**

```
EasycomConnection.EasycomConnection(System.String
sConnString)
EasycomConnection.EasycomConnection()
```

#### **Description**

Parameter	Description	Type
<code>connectionString</code>	Specifies the connection string, including the necessary parameters to connect using Easycom. See EasycomConnection.ConnectionString property for options available in this connection string.	String

You need to explicitly open and close connections using the Open and Close methods, respectively.

Another way to instantiate an EasycomConnection object is to instantiate an [EacConnection](#) object (because EacConnection derives from EasycomConnection).

Because this connection can be "pooled", when you close the object (using .Dispose() or .Close() method) the physical connection may remain open.

#### **Example of use (C#)**

```
EasycomConnection my_cnx;

my_cnx = new EasycomConnection();
my_cnx.ConnectionString = "Server=194.206.160.111;User. "
my_cnx.Open();
```

#### **Example of use (VB)**

```
Dim my_cnx As EasycomConnection
```

```
my_cnx = New EasycomConnection  
my_cnx.ConnectionString = "Server=194.206.160.111;User"  
my_cnx.Open()
```

### *EasycomConnection.ConnectionString Property*

Represents all options for the connection.

**Class** `EasycomConnection`

#### **Syntax**

```
public property String ConnectionString
```

#### **Description**

You set the ConnectionString property before opening a connection.

The option names are the same that the options available for `EasycomConnection.ConnectionString`:

- **Server:** The name or the IP address of the AS/400
- **User ID:** The user profile. In interactive environment, if this parameter is omitted an Easycom dialog box will be shown. If this parameter is omitted in a service (for example in ASP.NET environment) an exception will be thrown during the connection.
- **Password:** The password associated with the username.
- **JobName:** The job name to use. By default this is the name of the PC.
- **Pooled:** Choose whether or not the connection object will be part of a connection pool. See 'Pooling connection' topic for more details. Possible values are 'True' and 'False'. Default is True.
- **PoolLimit:** This value is an integer representing the maximum number of connections being pooled. When the limit is reached the thread will wait until a pooled connection is available or until the timeout is reached (see PoolTimeout parameter and 'Pooling connections' topic for more details). Default is 0 (unlimited). **Windows only.**
- **PoolTimeout.** Represents the maximum number of seconds to wait if the pool limit is reached. Default is 0 (unlimited). **Windows only.**
- **Shared:** Choose if sharing the connection for the same thread. Possible values are 'True' and 'False'. Default is False.
- **Init Libl:** Libraries to be added to the default library list (that belong to the user ID). This is a semicolon-separated list, like "MYLIB1;MYLIB2".
- **EacUnLock:** A special password that can be validated by a user defined AS/400 program. This prevents non-authorized developers to use Easycom. See 'Advanced security features' in the server documentation.
- **CodepageFile:** designates an ASCII-EBCDIC translation file. This is mostly useful for DBCS countries like China, Japan, Korea.
- **ShortFieldNames:** tells Easycom to use only short forms for field names. This will ignore aliases of columns. Valid values are 'True' and 'False'. Default is False.
- **SqlNaming:** Choose between '.' and '/' to separate files and libraries. When using '.' the library list is not available (and current library is the name of the user). Possible values are '/' and '.'. Default is '/.

- **CommandTimeout:** The CommandTimeout is configurable for a given connection.  
If the timeout is reached for a command, the following Easycom exception is raised:  
{ "Easycom error, Generic error: Timeout has been reached." }

### *EasycomConnection.Open Method*

Opens the connection.

#### **Class** EasycomConnection

#### **Syntax**

```
public void EasycomConnection.Open;
```

#### **Description**

Call this method to open the connection. If the connections are pooled, this may only reuse an available connection.

If the EasycomConnection.ConnectionString is empty, the connection will open with all default values, and show the Easycom dialog box for the user and password (If being in service mode, like ASP.NET, an exception will be thrown with wrong user or password).

A connection is always materialized by a JOB on the AS/400. The Easycom jobs are running by default in the Easycom subsystem. Those jobs can be visualized using the WRKACTJOB SBS(EASYCOM) command on a AS/400 terminal emulation (5250).

However, a single job can correspond to several EasycomConnection objects, depending on the EasycomConnection.ConnectionString parameters.

### *EasycomConnection.OpenFile Method*

Opens a file.

#### **Class** EasycomConnection

#### **Syntax**

```
public EasycomFile EasycomConnection.OpenFile(String  
FileName)  
  
public EasycomFile EasycomConnection.OpenFile(String  
FileName, openFileAccess openMode)  
  
public EasycomFile EasycomConnection.OpenFile(String  
FileName, openFileAccess openMode, Boolean bCommit)
```

#### **Description**

Opens a physical or logical file on the server, this file can be natively used (ReadKey, ReadFirst, ...).

Parameter	Description	Type	Default Value
<code>FileName</code>	Specifies the name of the file to open. This filename must specify a name and can specify a library and/or a member.	String	N/A
<code>openMode</code>	Specifies the open mode (ReadOnly, ReadWrite, ExclusiveRead or ExclusiveReadWrite)	<code>openFileAccess</code>	ReadOnly
<code>bCommit</code>	If true, the file will be opened in journaled mode, and will take part of transactions.	Boolean	False

The filename can have the following forms:

- LIBRARY/FILE: the file is defined in an absolute form.
- FILE: the file will be opened using the \*LIBL (library list) rules.
- LIBRARY/FILE(MEMBER): the file is defined in an absolute form, with a specific member (allows multi-member capable programs).
- FILE(MEMBER): opens a file using the \*LIBL rules, with a specific member.
- \*PGM/PGM\_DEF: opens an Easycom-defined program description. Allows native program calls (using key search function).

If the member is not specified, the default member is opened. If a file has no member the function will fail.

This function returns an instance to **EasycomFile** class which is designed to allow native actions possible to a file, including:

- key access
- direct update/insert/delete
- record locking
- native information on fields (column heading, AS/400 data type, lengths, etc.)
- saving/retrieving position (using Relative Record Numbers)

Remarks:

- the LIBRARY can be QTEMP. But be careful: QTEMP is a "memory" library valid only for the current JOB
- the file opening process will follow file substitutions that are performed on the job scope (OVRSCOPE(\*JOB) parameter of OVRDBF AS/400 command).

#### See also

EasycomFile constructor

#### Example (C#):

```
EasycomFile ecfile = new EasycomFile (Cnx, "EASYCOMXMP/SP_CUST",  
openFileAccess.ReadWrite);
```

### *EasycomConnection.RemoteCommand Method*

Executes a native command.

**Class** EasycomConnection

#### **Syntax**

```
public void RemoteCommand(String strCommand)
```

#### **Description**

EasycomConnection.RemoteCommand executes a command for the current connection. This command can affect the environment (JOB) of the current connection. For example, a command can run a program that will change the library list, the objects overrides, and so on.

An exception (EasycomException) will be thrown if the command failed (command syntax, or execution failure ...).

The command syntax is like AS/400 terminal command line (best is to test the command syntax in the terminal before using it in the method).

Restrictions: this method cannot call interactive commands (like DSPMSG). This method cannot call commands that require return variables: use the RemoteRtvCommand instead.

#### **Example (VB.NET):**

```
Try
    mycnx.RemoteCommand("ADDLIB MYLIB")
    mycnx.RemoteCommand('CHGDTAARA DTAARA(*LDA (1 2))
    VALUE(AA) ');
Catch except As Exception
    MessageBox.Show("Error while executing command : " +
    except.Message, Me.Text)
End Try
```

#### **Example (C#):**

```
try
{
    mycnx.RemoteCommand("ADDLIB MYLIB");
    mycnx.RemoteCommand("'CHGDTAARA DTAARA(*LDA (1 2))
    VALUE(AA) ");
}
catch (EasycomException exp)
{
    MessageBox.Show("Error while executing command : "+
    exp.Message);
}
```

### *EasycomConnection.RemoteRtvCommand Method*

Executes a native command which returns data.

**Class** EasycomConnection

### Syntax

```
public void RemoteRtvCommand(String strCommand)
```

### Description

EasycomConnection.RemoteRtvCommand executes a command on the current connection, with capability to have return values. The command syntax is like CL language.

This command can affect the environment (JOB) of the current connection or retrieve some values depending on the job or the system.

The strCommand syntax is like a regular CL syntax, with & as variables prefixes. For example:

RTVJOBA USER(&USR)

If a variable as a length superior to 20 characters or if a variable is not character type, you need to declare it with CHAR(len) or DEC(len dec). All declarations are semicolon-separated. For example:

LIBL=CHAR(2750);DFTWAIT=DEC(7 0);RTVJOBA USRLIBL(&LIBL)

Subsequent calls of EasycomConnection.RemoteRtvCommandGetValue function is used to get the returned variables.

An exception (EasycomException) will be thrown if the command syntax is incorrect. However, an exception is not thrown upon failure (as opposed to the EasycomConnection.RemoteCommand method).

If the command syntax was correct, but the execution failed, a special variable name "RC" will contain the CPF code of the error (no exception thrown). So, a typical use of this method will execute the command, test the "RC" variable, and finally get the other variable(s).

Remark: it is recommended to use RemoteCommand method if the command is not returning any values (better performance, and better error handling).

### Example of use (VB.NET):

```
Dim my_cnx As EasycomConnection
Dim result As String

my_cnx = New EasycomConnection
my_cnx.ConnectionString = "Server=194.206.160.111;User"
my_cnx.Open()
my_cnx.RemoteRtvCommand("LIBL=CHAR(2750);DFTWAIT=DEC(7 0);RTVJOBA USRLIBL(&LIBL)")
If my_cnx.RemoteRtvCommandGetValue("RC") = "0" Then
    MessageBox.Show("User Library List: " +
        my_cnx.RemoteRtvCommandGetValue("LIBL"))
    MessageBox.Show("User Default Wait: " +
        my_cnx.RemoteRtvCommandGetValue("DFTWAIT"))
```

```
Else
    MessageBox.Show("The command call returned the
        following message" +
        my_cnx.RemoteRtvCommandGetValue("RC"))
End If
```

**Example of use (C#):**

```
EasycomConnection my_cnx = new EasycomConnection();
my_cnx.ConnectionString = "Server=194.206.160.111;User
Id=";
my_cnx.Open();
my_cnx.RemoteRtvCommand("LIBL=CHAR(2750);DFTWAIT=DEC(7
0);RTVJOBA USRLIBL(&LIBL)");
if( my_cnx.RemoteRtvCommandGetValue("RC") == "0")
{
    MessageBox.Show("User Library List: " +
        my_cnx.RemoteRtvCommandGetValue("LIBL"));
    MessageBox.Show("User Default Wait: " +
        my_cnx.RemoteRtvCommandGetValue("DFTWAIT"));
}
else
{
    MessageBox.Show("The command call returned the
        following message" +
        my_cnx.RemoteRtvCommandGetValue("RC"));
}
```

***EasycomConnection.RemoteRtvCommandGetValue Method***

Gets a result variable for the last call of RemoteRtvCommand call.

**Class EasycomConnection****Syntax**

```
public String RemoteRtvCommandGetValue(strVariable:
String)
```

**Description**

EasycomConnection.RemoteRtvCommandGetValue gets a result variable for the last call of RemoteRtvCommand.

A program should always get the "RC" variable before getting other variable(s).

**Example of use (C#):**

```
EasycomConnection my_cnx = new EasycomConnection();
my_cnx.ConnectionString = "Server=194.206.160.111;User
Id=";
my_cnx.Open();
my_cnx.RemoteRtvCommand("LIBL=CHAR(2750);DFTWAIT=DEC(7
0);RTVJOBA USRLIBL(&LIBL)");
if( my_cnx.RemoteRtvCommandGetValue("RC") == "0")
{
    MessageBox.Show("User Library List: " +
        my_cnx.RemoteRtvCommandGetValue("LIBL"));
}
```

```
        MessageBox.Show("User Default Wait: " +  
        my_cnx.RemoteRtvCommandGetValue("DFTWAIT"));  
    }  
    else  
    {  
        MessageBox.Show("The command call returned the  
        following message" +  
        my_cnx.RemoteRtvCommandGetValue("RC"));  
    }
```

The return type is always string, it does not depend on the datatype of the host variable. However, if the datatype of the return variable is decimal, it must be declared in the RemoteRtvCommand string.

See [EasycomConnection.RemoteRtvCommand](#) for more details.

## EasycomDataSet

### *EasycomDataSet Class*

Namespace **Easycom**

#### Syntax

```
public class EasycomDataSet : System.Object
```

#### Description

The EasycomDataSet class is the lowest-level class that handles basic operation on an opened file.

Those basic operations are: navigation (first, next, last, previous), getting data (by fieldname or by index), saving and restoring position, querying information on key or fields.

### *EasycomDataSet.GetFieldCount Method*

Returns the number of fields in the dataset.

Class **EasycomDataSet**

#### Syntax

```
public Integer EasycomDataSet.GetFieldCount()
```

#### Example of use (C#):

```
EasycomConnection my_cnx = new EasycomConnection();  
my_cnx.ConnectionString = "Server=194.206.160.111;User  
Id=";  
my_cnx.Open();  
EasycomFile ecfile = new EasycomFile(my_cnx, filepath,  
OpenFileAccess.ReadWrite);  
int nb = ecfile.GetFieldCount();
```

This function returns the total number of fields in the dataset.



### See also

[EasycomDataSet.GetFieldInformation](#)

### *EasycomDataSet.GetFieldInformation Method*

Returns information on a specified field.

### Class EasycomDataSet

### Syntax

```
public Object GetFieldInformation(Integer f,
    Easycom.fieldInformation info)

public Object GetFieldInformation(String fieldName,
    Easycom.fieldInformation info)
```

### Description

This function returns the requested information as an object, which type depends on the information being requested.

fieldInformation FieldName	Type String	Comments The field name, short or long (10 or 30).
EasycomType	EasycomType	The native datatype
Digits	Short	Total number of digits
Decimals	Short	Number of decimals
ServerLen	Short	Field size, in bytes
ColumnHeader	String	The AS/400 column header, 60 chars.
ColumnText	String	The AS/400 column text, 50 chars
Nullallowed	Boolean	True if the field can be null

### Example of use (C#):

```
EasycomConnection my_cnx = new EasycomConnection();
my_cnx.ConnectionString = "Server=194.206.160.111;User
Id=";
my_cnx.Open();
EasycomFile ecfile = new EasycomFile(my_cnx, filepath,
OpenFileAccess.ReadWrite);
EasycomType type =
(EasycomType)ecfile.GetFieldInformation(fieldnumber,
Easycom.fieldInformation.EasycomType));
```

### See also

[EasycomDataSet.GetFormatInformation](#)

### *EasycomDataSet.GetFormatInformation Method*

Returns information on the opened dataset.

## Class EasycomDataSet

### Syntax

```
public Object
GetFormatInformation(Easycom.formatInformation info)
```

### Description

This function returns the requested information as an object, which type depends on the information being requested.

formatInformation	Type	Comments
fieldCount	Int	The total number of fields
fldKeyCount	Short	The number of fields in the key
keyUnique	Boolean	True if the key is unique
keySelOmit	Boolean	True if the key has a select or omit clause

### See also

[EasycomDataSet.GetFieldInformation](#)

### *EasycomDataSet.GetPosition Method*

Gets the current position in the dataset.

## Class EasycomDataSet

### Syntax

```
public Integer EasycomDataSet.GetPosition(): Integer;
```

### Description

EasycomDataSet.GetPosition returns the current position.

If the dataset belongs to a file the position is a unique integer that is actually the AS/400 relative record number, which has no relationship with the record rank.

If the dataset is a query, the position is a value relative to the first or to the last record. The value is positive if the position is relative to the first record (for example = 4 if this is the 4<sup>th</sup> record), and negative if the position is relative to the last record (for example = -4 if this is the 4<sup>th</sup> record starting from last).

If there is no current position, an exception will be thrown (will occur if the position is asked just after having opened a file).

### Example of use (VB.NET):

The following example gets the position of the last record (that is not the rank of the last record).

```
Dim my_cnx As EasycomConnection
Dim my_file As EasycomFile
Dim posi As Integer

my_cnx = New EasycomConnection
my_cnx.ConnectionString = "Server=194.206.160.111;User Id=
my_cnx.Open()

my_file = my_cnx.OpenFile("S_CUSTOMER")

If my_file.ReadLast Then
posi = my_file.GetPosition()
End If
```

**Example of use (C#) :**

The following example gets the position of the last record (that is not the rank of the last record).

```
EasycomConnection my_cnx = new EasycomConnection();
EasycomFile my_file = new
EasycomFile(my_cnx, "S_CUSTOMER");
int position =0;

my_cnx.ConnectionString = "Server=194.206.160.111;User
Id=";
my_cnx.Open();
my_file = my_cnx.OpenFile("S_CUSTOMER")

if (my_file.ReadLast())
{
    position = my_file.GetPosition();
}
```

***EasycomDataSet.GotoPosition Method***

Sets the current position in the dataset.

**Class EasycomDataSet****Syntax**

```
public void EasycomDataSet.GotoPosition(Integer pos)
```

**Description**

EasycomDataSet.GotoPosition reads the record at the given position.

If the dataset belongs to a file the position is a unique integer that is the AS/400 relative record number, which has no relationship with the record rank.

If the dataset is a query, the position is a value relative to the first or to the last record. The value is positive if the position is relative to the first record (for example = 4 if this is the 4<sup>th</sup> record), and negative if the position is relative to the last record (for example = -4 if this is the 4<sup>th</sup> record starting from last).

If the dataset is an EasycomFile object, the fetched record may be locked or not depending on the current fetch mode. The fetch mode is sequential read with no lock by default and can be changed using the EasycomFile.SetFetchmode method.

If the position is unreachable an exception will be thrown.

#### See also

[EasycomDataSet.GetValue](#)

[EasycomFile.SetFetchmode](#)

### *EasycomDataSet.GetValue Method*

Get field values.

#### Class EasycomDataSet

#### Syntax

```
public Object EasycomDataSet.GetValue(Integer f)
public Object EasycomDataSet.GetValue(String flfName)
```

#### Description

These functions return the field value as an object. The object will be the nearest object type that corresponds to the database type of the field.

The Object can be a String, Integer, SmallInt, ...

The value can be obtained from the field name or from the field index (zero-based). The field name is the alias name of the field if existing, except if short field names are forced using the connection string options.

#### Example of use (VB.NET)

```
Dim my_cnx As EasycomConnection
Dim my_file As EasycomFile
Dim result As String

my_cnx = New EasycomConnection
my_cnx.ConnectionString = "Server=194.206.160.111;User
Id=
my_cnx.Open()
my_file = my_cnx.OpenFile("S_CUSTOMER")
```

```
my_file.ReadFirst()  
result = my_file.GetValue("COMPANY").ToString() + " - " +  
my_file.GetValue("FIRSTNAME").ToString()  
MessageBox.Show("Customer found : " + result)
```

**Example of use (C#):**

```
EasycomConnection my_cnx = new EasycomConnection();  
my_cnx.ConnectionString = "Server=194.206.160.111;User  
Id=";  
my_cnx.Open();  
EasycomFile ecfile = my_cnx.OpenFile("S_CUSTOMER");  
Ecfile.ReadFirst();  
string result = ecfile.GetValue("COMPANY").ToString() + "  
- " + ecfile.GetValue("FIRSTNAME").ToString();  
MessageBox.Show("Customer found : " + result);
```

**See also**[EasycomDataSet.ReadFirst](#)[EasycomDataSet.ReadNext](#)[EasycomDataSet.IsDBNull](#)***EasycomDataSet.IsDBNull Method***

Tests null value state of a field.

**Class EasycomDataSet****Syntax**

```
public Boolean EasycomDataSet.IsDBNull(Integer f)
```

**Description**

This function returns true if the field value is null.

**See also**[EasycomDataSet.ReadFirst](#)[EasycomDataSet.ReadNext](#)[EasycomDataSet.GetValue](#)***EasycomDataSet.Readxxx Methods***

Reads a record

**Class EasycomDataSet****Syntax**

```
public Boolean EasycomDataSet.ReadFirst()  
public Boolean EasycomDataSet.ReadNext()  
public Boolean EasycomDataSet.ReadPrev()  
public Boolean EasycomDataSet.ReadLast()
```

## Description

These functions are reading a record using the given direction. The function returns true if a record is fetched and false otherwise.

If there is an EasycomFile object, the fetched record may be locked or not depending on the current fetch mode. The fetch mode is sequential read with no lock by default and can be changed using the EasycomFile.SetFetchmode method.

The field values for the current record can be accessed using the GetValue function.

## Example of use (C#)

```
EasycomConnection my_cnx;
EasycomFile my_file;
string result;

my_cnx = new EasycomConnection();
my_cnx.ConnectionString = "Server=194.206.160.111;User Id=
my_cnx.Open();
my_file = my_cnx.OpenFile("S_CUSTOMER");
my_file.ReadFirst();
result = my_file.GetValue("COMPANY").ToString() + " - " +
my_file.GetValue("FIRSTNAME").ToString();
MessageBox.Show("Customer found : "+result);
while (my_file.ReadNext())
{
    result += my_file.GetValue("COMPANY").ToString() + " - "
+ my_file.GetValue("FIRSTNAME").ToString();
}
```

## Example of use (VB.NET)

```
Dim my_cnx As EasycomConnection
Dim my_file As EasycomFile
Dim result As String

my_cnx = New EasycomConnection
my_cnx.ConnectionString = "Server=194.206.160.111;User Id=
my_cnx.Open()
my_file = my_cnx.OpenFile("S_CUSTOMER")
my_file.ReadFirst()
result = my_file.GetValue("COMPANY").ToString() + " - " +
my_file.GetValue("FIRSTNAME").ToString()
MessageBox.Show("Customer found : " + result)
while (my_file.ReadNext())
{
    result += my_file.GetValue("COMPANY").ToString() + " - "
+ my_file.GetValue("FIRSTNAME").ToString()
}
```

**See also**[EasycomDataSet.GetValue](#)[EasycomFile.SetFetchmode](#)**EasycomFile****EasycomFile Class****Namespace** Easycom**Syntax**

```
public class EasycomFile : EasycomDataSet
```

**Description**

The EasycomFile is a class that handles all possible native operations on an AS/400 file.

Basic operations are available on the parent class "EasycomDataSet".

File specific operations are available on this class, including record-locking, insert, update, delete, key seek.

**EasycomFile Constructor**

Creates a new instance of a EasycomFile object, which represents an opened file in an AS/400 connection.

**Class EasycomFile****Syntax**

```
public EasycomFile.EasycomFile(EasycomConnection cnx,  
String FileName)  
public EasycomFile.EasycomFile(EasycomConnection cnx,  
String FileName, openFileAccess openMode);  
public EasycomFile.EasycomFile(EasycomConnection cnx,  
String FileName, openFileAccess openMode, Boolean  
bCommit)
```

**Description**

Parameter	Description	Type
<a href="#">Cnx</a>	Specifies the connection object to which the file opening will belong to	Easycomconnction
<a href="#">FileName</a>	Specifies the name of the file to open. This filename must specify a name, and can specify a library and/or a member.	String
<a href="#">openMode</a>	Specifies the open mode (ReadOnly, ReadWrite, ExclusiveRead or	openFileAccess

ExclusiveReadWrite)		
bCommit	If true, the file will be opened in journaled mode, and will take part of transactions.	Boolean

The filename can have the following forms:

- LIBRARY/FILE: the file is defined in an absolute form.
- FILE: the file will be opened using the \*LIBL (library list) rules.
- LIBRARY/FILE(MEMBER): the file is defined in an absolute form, with a specific member (allows multi-member capable programs).
- FILE(MEMBER): opens a file using the \*LIBL rules, with a specific member.
- \*PGM/PGM\_DEF: opens an Easycom-defined program description. Allows native program calls (using key search function).

### *EasycomFile.Readxxx Method*

Reads a record

#### **Class EasycomFile**

##### **Syntax**

```
public Boolean EasycomFile.ReadFirst()  
public Boolean EasycomFile.ReadNext()  
public Boolean EasycomFile.ReadPrev()  
public Boolean EasycomFile.ReadLast()
```

##### **Description**

These functions are reading a record using the given direction. The function returns true if a record is fetched and false otherwise.

See also [EasycomDataSet.Readxxx](#) methods

### *EasycomFile.ReadKey Method*

Seeks a record with key values

#### **Class EasycomFile**

##### **Syntax**

```
public Boolean EasycomFile.ReadKey(object keyValues)  
public Boolean EasycomFile.ReadKey object keyValues,  
Integer lastFieldPartialLen)  
public Boolean EasycomFile.ReadKey(object keyValues,  
keyReadOp op)  
public Boolean EasycomFile.ReadKey(object keyValues,  
keyReadOp op, Integer lastFieldPartialLen)
```

##### **Description**

These functions seek a record using the specified values for the key. The function returns true if the record was found and false otherwise.



Parameter	Description	Type
<code>keyValues</code>	Specifies the key values. If there is only one key field or if the seek must be done only on the first field, the value can be a simple CLR type (like string, integer, ...). If several key values need to be specified, the keyValues object must be an array of CLR types.	object
<code>lastFieldPartialLen</code>	Allows partial key search. This represents the length, in bytes, of the significant part for the last key value specified.	Integer
<code>Op</code>	Specifies the key operation verb. The possible values are : <ul style="list-style-type: none"><li>• Eq: must be equal (totally or partially depending on other parameters).</li><li>• Ge: must be greater or equal (<math>\geq</math>)</li><li>• Le: must be less or equal (<math>\leq</math>)</li><li>• Gt: must be strictly greater (<math>&gt;</math>)</li><li>• Lt: must be strictly less (<math>&lt;</math>)</li><li>• NextEq: must follow current record and be equal</li><li>• PrevEq: must precede current record and be equal</li><li>• NextUnEq: must follow current record and be not equal</li><li>• PrevUnEq: must precede current record and be not equal</li></ul>	keyRead

This function allows real native usage of the key that is in the logical or physical file. The application must open the correct logical file in order to use a specified key (then it is possible to use `EasycomDataSet.GetPosition/EasycomDataSet.GotoPosition` to synchronise different logical files).

The "lastFieldPartialKeyLen" allows partial key search. If n key values are specified in the keyValues parameters, "lastFieldPartialKeyLen" represents the size of the n<sup>th</sup> key field. In other words, the key search will ignore the rest of the key.

If "lastFieldPartialKeyLen" is omitted the read will be performed on all key fields specified (ignoring others).

### Examples (VB.NET)

Suppose that you have a logical file that has a key with Country ID has Packed Decimal 4 digits, and customer name. If you want to seek the first customer that begins with "LA" string in a specified country, you will have a code like:

```
Dim my_cnx As EasycomConnection
Dim my_file As EasycomFile
Dim result As String
Dim s_values() As Object = {"US", "P"}

my_cnx = New EasycomConnection
my_cnx.ConnectionString = "Server=194.206.160.111;User Id=
my_cnx.Open()

my_file = my_cnx.OpenFile("s_cust_cnfn")
    ' A search with a compound key
If my_file.ReadKey(s_values, 1) Then
    MessageBox.Show("Customer ID: " +
my_file.GetValue("CUST_ID").ToString())
    MessageBox.Show("Customer Name: " +
my_file.GetValue("FIRSTNAME").ToString())
    MessageBox.Show("Country: " +
my_file.GetValue("COUNTRY").ToString())
End If

'To seek the first "US" country:
my_file.ReadKey("US")

'To seek first customer beginning with "Jo":
my_file = my_cnx.OpenFile("s_cust_na")
my_file.ReadKey("Jo", 2)
```

### Examples (C#)

Suppose that you have a logical file that has a key with Country ID has Packed Decimal 4 digits, and customer name. If you want to seek the first customer that begins with "LA" string in a specified country, you will have a code like:

```
EasycomConnection my_cnx;
EasycomFile my_file;
string result;
object s_values[] = {"US", "P"}

my_cnx = new EasycomConnection();
my_cnx.ConnectionString = "Server=194.206.160.111;User Id=
my_cnx.Open();

my_file = my_cnx.OpenFile("s_cust_cnfn");
    ' A search with a compound key
if my_file.ReadKey(s_values, 1)
{
    MessageBox.Show("Customer ID: " +
my_file.GetValue("CUST_ID").ToString());
    MessageBox.Show("Customer Name: " +
my_file.GetValue("FIRSTNAME").ToString());
}
```

```
MessageBox.Show("Country: " +  
my_file.GetValue("COUNTRY").ToString());  
}  
  
'To seek the first "US" country:  
my_file.ReadKey("US");  
  
'To seek first customer beginning with "Jo":  
my_file = my_cnx.OpenFile("s_cust_na");  
my_file.ReadKey("Jo", 2);
```

**See also**[EasycomDataSet.GetPosition](#)[EasycomDataSet.GotoPosition](#)***EasycomFile.RecordAppend Method***

Puts the EasycomFile object into an append mode.

**Class EasycomFile****Syntax**

```
public void EasycomFile.RecordAppend(Boolean clear)  
public void EasycomFile.RecordAppend()
```

**Description**

This function puts the EasycomFile object into append mode. If clear is true (default), all fields will be cleared. Otherwise, the field values of the current record (or the last inserted record) will be used.

Subsequent calls to EasycomFile.SetValue must be performed to give field values.

A call to EasycomFile.RecordUpdate must be performed to validate the record appending.

***EasycomFile.RecordDelete Method***

Deletes the current record.

**Class EasycomFile****Syntax**

```
public void EasycomFile.RecordDelete()
```

**Description**

This function deletes the current record. The record must have been locked before (using the RecordLock method or using the Locked fetch mode).



### *EasycomFile.RecordLock Method*

Locks the current record.

**Class** EasycomFile

#### **Syntax**

```
public void EasycomFile.RecordLock()
```

#### **Description**

This function locks the current record. If the fetch mode is "Locked", this function has no effect.

If the record has changed between the fetch (ReadKey, ...) and the RecordLock function call, an exception will be thrown.

When the record is locked, EasycomFile.RecordDelete and EasycomFile.RecordUpdate are allowed.

### *EasycomFile.RecordUnLock Method*

Unlocks the current record.

**Class** EasycomFile

#### **Syntax**

```
public void EasycomFile.RecordUnLock()
```

#### **Description**

This function unlocks the current record. If the fetch mode is "Locked", this function unlocks the current record, but will continue to lock records for future fetches.

### *EasycomFile.RecordUpdate Method*

Validates record changes or record appending.

**Class** EasycomFile

#### **Syntax**

```
public void EasycomFile.RecordUpdate()
```

#### **Description**

This function validates record changes or record appending.

This function call is valid after a call of EasycomFile.Append, after a call of EasycomFile.RecordLock or if a record is fetched in "Locked" fetch mode. In other words, the object must be in append or in locked mode.

Previous calls of [EasycomFile.SetValue](#) must have been performed to set field values.

### *EasycomFile.SetFetchMode Method*

Chooses a fetch mode depending on the EasycomFile object usage needed.

#### **Class EasycomFile**

##### **Syntax**

```
public void EasycomFile.SetFetchMode(fetchMode mode)
```

##### **Description**

This function chooses a fetch mode. Possible values for mode are:

Locked: each fetch is locked. This fetches records only one by one, but always locked. This allows safe multiple updates.

Unlocked: standard native fetches. Limited cache size for optimum native access (key searches, ...)

SequentialUnlocked (default): Unlocked, but with large cache size. Optimized for Forward-only fetches.

### *EasycomFile.SetValue Method*

Defines a field value for modifying or inserting a record.

#### **Class EasycomFile**

##### **Syntax**

```
public void EasycomFile.SetValue(Integer f, Object Value)
public void EasycomFile.SetValue(String fieldName, Object Value)
```

##### **Description**

This function changes a value for the specified field. This function call is valid only when having the current record locked or when being in append mode.

#### **Example 1 (VB.NET):**

This example uses the implicit record locking. Lock error may occur when reading the record.

```
Dim my_cnx As EasycomConnection
Dim my_file As EasycomFile

my_cnx = New EasycomConnection
my_cnx.ConnectionString = "Server=194.206.160.111;User
Id=
```

```
my_cnx.Open()

my_file = my_cnx.OpenFile("s_customer",
openFileAccess.ReadWrite)
my_file.SetFetchMode(fetchMode.Locked)

If my_file.ReadKey(1354) Then
my_file.SetValue("STATE", "CA")
my_file.SetValue("CITY", "San Francisco")
my_file.RecordUpdate()
End If
```

### Example 2 (VB.NET):

This example uses the explicit record locking. Lock error may occur when locking record, if lock fails or if the record changed between the "read" and the "lock".

```
Dim my_cnx As EasycomConnection
Dim my_file As EasycomFile

my_cnx = New EasycomConnection
my_cnx.ConnectionString = "Server=194.206.160.111;User
Id=
my_cnx.Open()

my_file = my_cnx.OpenFile("s_customer",
openFileAccess.ReadWrite)

If my_file.ReadKey(1354) Then
my_file.RecordLock()
my_file.SetValue("STATE", "CA")
my_file.SetValue("CITY", "San Francisco")
my_file.RecordUpdate()
End If
```

### Example 1 (C#):

This example uses the implicit record locking. Lock error may occur when reading the record.

```
EasycomConnection my_cnx;
EasycomFile my_file;

my_cnx = new EasycomConnection();
my_cnx.ConnectionString = "Server=194.206.160.111;User Id=
my_cnx.Open();
```



```
my_file = my_cnx.OpenFile("s_customer",
    openFileAccess.ReadWrite);
my_file.SetFetchMode(fetchMode.Locked);

if (my_file.ReadKey(1354))
{
    my_file.SetValue("STATE", "CA");
    my_file.SetValue("CITY", "San Francisco");
    my_file.RecordUpdate();
}
```

### Example 2 (C#);

This example uses the explicit record locking. Lock error may occur when locking record if lock fails or if the record changed between the "read" and the "lock".

```
EasycomConnection my_cnx;
EasycomFile my_file;

my_cnx = new EasycomConnection();
my_cnx.ConnectionString = "Server=194.206.160.111;User
Id=
my_cnx.Open();

my_file = my_cnx.OpenFile("s_customer",
    openFileAccess.ReadWrite);

if (my_file.ReadKey(1354))
{
    my_file.RecordLock();
    my_file.SetValue("STATE", "CA");
    my_file.SetValue("CITY", "San Francisco");
    my_file.RecordUpdate();
}
```

### Example

```
EasycomConnection ConnectionIBM new EasycomConnection();

ConnectionIBM .ConnectionString = "Server=" + textBoxServer.Text +
";User Id=" + textBoxUser.Text + ";Password=" + textBoxPwd.Text +
";Pooled=False";

ConnectionIBM.Open();

EasycomFile ecfile = new EasycomFile(ConnectionIBM,
    "EASYCOMXMP/SP_CUST", openFileAccess.ReadWrite);

ecfile.ReadFirst();
```

```
string result = ecfile.GetValue("LASTNAME").ToString() + " - " +  
ecfile.GetValue("FIRSTNAME").ToString();  
MessageBox.Show("Customer found : " + result);
```

## EacXML

### *EacXML Class*

**Namespace** Easycom

#### Syntax

```
public class EacXML : Eac32vbaWrapper
```

#### Description

EacXML is an interface to describe and call native IBM i Procedure and Programs, using an XML protocol. A set of API functions is provided:

- xmlDefine
- xmlLoadDefinition
- xmlBindSrvPgm
- xmlCallProgram
- xmlExecRequest

Original RPG sources and/or PCML can be used by .NET programs to describe data structures, procedures, and programs.

### *EacXML Constructor*

Creates a new instance of an EacXML object, which represents a program call.

**Class** EacXML

#### Syntax

```
public EacXML.EacXML(EacConnection cnx);  
public public EacXML.EacXML (EasycomConnection cnx);
```

#### Description

Parameter	Description	Type
Cnx	Specifies the connection object	EasycomConnection or



---

to which the program opening  
will belong to

---

EacConnection

The connection has to be valid and open.

### *EacXML.XMLDefine Method*

Asks Easycom to load the description of a program, the program's interface can be described using simplified RPG or PCML. It returns true if the definition loads normally and throw an EasycomException it case there was a problem.

#### **Class EacXML**

##### **Syntax**

```
public bool EacXML.XMLdefine(string DescriptionType,  
string Description);
```

##### **Description**

Parameter	Description	Type
<code>DescriptionType</code>	Specifies the type of description we will be using; the only possible values are RPG and PCML	String
<code>Description</code>	Specifies the connection object to which the program opening will belong to	String

The connection must be valid and open.

### *EacXML.XMLLoadDefinition Method*

Asks Easycom to load the description of a program, the program's interface can be described using simplified RPG or PCML. It returns true if the definition loads normally and throw an EasycomException it case there was a problem.

#### **Class EacXML**

##### **Syntax**

```
public bool EacXML.XMLdefine(string DescriptionType,  
string DescriptionFile);
```



Description		
Parameter	Description	Type
<code>DescriptionType</code>	Specifies the type of description we will be using; the only possible values are RPG and PCML	String
<code>DescriptionFile</code>	Specifies the location of the file containing the description. The file can either be on the IFS or the native IBM I system	String

The connection must be valid and open.

#### *EacXML.XMLBindSrv Method*

Asks Easycom to bind a service program, functions from that service program can then be called if their definition has been loaded. It returns true if the binding is successful and throw an EasycomException it case there was a problem.

#### **Class** `EacXML`

#### **Syntax**

```
public bool EacXML.XMLBindSrv(string ServiceProgram);
```

#### **Description**

Parameter	Description	Type
<code>ServiceProgram</code>	Specifies the name of the service program to load	String

The connection has to be valid and open.

#### *EacXML.XMLCallProgram Method*

Asks Easycom to call the program and return the results. You must first load the definition of the program using XmlDefine or XMLLoadDefinition.

The function returns the result of the program call under an XML format and throws an EasycomException it case there was a problem.

The input parameters also have to be under an XML format.

**Class EacXML****Syntax**

```
public string EacXML.XMLcallProgram(string ProgramName,  
string InputParameters);
```

**Description**

Parameter	Description	Type
ProgramName	Specifies name of the program to call	String
InputParameters	Specifies the input parameters as an XML string.	String

The connection has to be valid and open.

***EacXML.XMLExecRequest Method***

Asks Easycom to call the program defined within the input String parameter called "Request". The "Request" parameter contains the XML input file content.

Indeed, the input parameter must be under an XML format.

The function returns the result of the program call under an XML format and throws an EasycomException if case there was a problem.

**Class EacXML****Syntax**

```
public string EacXML.XMLExecRequest(string Request);
```

**Description**

Parameter	Description	Type
Request	Specifies request	String

The connection has to be valid and open.

***Example 1***

Into this example, we use a RPG description:

```
EacXML oXml = new EacXML(ConnectionIBM);
```



```
oXml.XmlBindSrv("XMPSRVPGM");
oXml.XmlLoadDefinition("RPG", "EASYCOMXMP/QRPGLESRC,CVTNW_H");

string result = oXml.XmlCallProgram("FCVTNW",
"<LIMIT>150</LIMIT><DECV>12345</DECV>");

result = result.Substring(0, result.IndexOf("</ReturnValue>") + 14);

XmlDocument doc = new XmlDocument();
doc.LoadXml(result);
XmlNode infos = doc.SelectSingleNode("ReturnValue");
string content = infos.InnerText;
MessageBox.Show(content);
```

### Example 2

Into this example, we use a PCML description, passed as string parameter of the XmlDefine method:

```
EacXML myEacXML = new EacXML(ConnectionIBM);

myEacXML.XmlDefine("PCML", " <pcml version = \"4.0\"><program name = \"QXXRTVDA\"
path = \"/QSYS.lib/QXXRTVDA.pgm\"><data name = \"DTANAME\" type = \"char\"
length = \"20\" usage = \"input\" /><data name = \"OFFSET\" type = \"int\"
usage = \"input\" length = \"4\" /><data name = \"DATALEN\" type = \"int\"
usage = \"input\" length = \"4\" /><data name = \"RESULT\" type = \"int\" length = \"4\"
usage = \"output\" /></program></pcml>");

myEacXML.XmlCallProgram("QXXRTVDA", "<DTANAME>CFGECAC *LIBL
</DTANAME><OFFSET>29</OFFSET><DATALEN>4</DATALEN>");
```

### Example 3

This example is provided with [demo samples](#).

PCML descriptions are used. PCML file is on Windows side or on IBMi IFS.

```
EacXML myEacXML = new EacXML(ConnectionIBM);

//Load program prototypes
XDocument doc = XDocument.Load("define.xml");

string stringXMLOut = myEacXML.XmlExecRequest(XMLToString(doc));

myEacXML.XmlLoadDefinition("PCML",
"/usr/local/easycom/PCML/QSYRUSRI.pcm1");

// Call QGYOLAUS system API
doc = XDocument.Load("program.xml");
stringXMLOut = myEacXML.XmlExecRequest(XMLToString(doc));

//Parse XML out to retrieve the following values:totalRcds,
rcdLength, rqsHandle
XElement root =
XElement.Parse(stringXMLOut).Element("Program").Element("ParameterLi
st").Element("listInfo");
```



```
string requestHandle = (string)root.Element("rqsHandle");
int rcdsToReturn = (int)root.Element("totalRcds");
int rcdLength = (int)root.Element("rcdLength");
int receiverLength = rcdLength * rcdsToReturn;

// Call qgygt1 program
doc = XDocument.Load("program2.xml");
XElement Elmt = doc.Root.Element("Program");

Elmt.Element("parameterList").Add(new XElement("receiverLength",
receiverLength.ToString()));
Elmt.Element("parameterList").Add(new XElement("requestHandle",
requestHandle));
Elmt.Element("parameterList").Add(new XElement("rcdsToReturn",
rcdsToReturn.ToString()));
Elmt.Element("parameterList").Add(new XElement("STARTINGRCD", "1"));

stringXMLOut = myEacXML.XmlExecRequest(XMLToString(doc));

//Parse XML out to create list of Users. Retrieve the following
values: NAME
XElement root2 = XElement.Parse(stringXMLOut);
List<string> userList = new List<string>();

IEnumerable<XElement> item2 =
    from el in
        root2.Element("Program").Element("ParameterList").Element("receiver
").Elements("item")
        select el.Element("NAME");

foreach (XElement el in item2)
{
    userList.Add(el.Value);
}

BindingSource bs = new BindingSource();
bs.DataSource = userList;
checkedListBoxUsers.DataSource = bs;
```

define.xml :

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?><Easycom
LogFile="/tmp/xml_demo_cedrick.log" DebugLog="259">
    <define>
        <file type="PCML" stmf="/usr/local/easycom/PCML/QGYOLAUS.pcm1" />
    </define>
</Easycom>
```

program.xml :

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?><Easycom
LogFile="/tmp/xml_demo_cedrick.log" DebugLog="259">
    <Program Name="QGYOLAUS">
        <parameterList></parameterList>
    </Program>
</Easycom>
```

program2.xml:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?><Easycom
LogFile="/tmp/xml_demo_cedrick.log" DebugLog="259">
  <Program Name="qgygtle">
    <parameterList></parameterList>
  </Program>
</Easycom>
```

#### Example 4

PCML decription is used. The PCML file is located on IBMi IFS.

```
EacXML oXml = new EacXML(ConnectionIBM);
oXml.XmlLoadDefinition("PCML", "/tmp/rpcsample.pcm1");
string result = oXml.XmlCallProgram("RPCSAMPLE",
                                     "<OP1>2</OP1><OP2>3</OP2><STR1>Aura</STR1>");

//string result contains:
//< ParameterList >
//< OP2 Type = "Packed" > 5.00 </ OP2 >
//< STR2 Type = "Char" > Aura </ STR2 >
//< OP3 Type = "Packed" > 6.0000 </ OP3 >
//</ ParameterList >

XElement root = XElement.Parse(result);

string displayResult = "";
foreach (XElement el in root.Descendants())
{
    displayResult = displayResult + el.Name + " = " + el.Value + " - ";
}
MessageBox.Show(displayResult);
```

#### Example 5

This is a simple RPG program:

```
D  PARM1      S      10a
D  PARM2      S      20a

C  *entry      plist
C          PARM      PARM1
C          PARM      PARM2

/free
  PARM2 = PARM1;
  return;

/end-free
```

This RPG has been compiled using these commands:

```
CRTRPGMOD MODULE(EASYCOMXM3/SAMPLEPG1) SRCFILE(EASYCOMXM3/QRPGLESRC)
SRCMBR(SAMPLEPG1) PGMINFO(*PCML) INFOSTMF('/tmp/SAMPLEPG1_easycomxm3.pcml')
```

```
CRTSRVPGM SRVPGM(EASYCOMXM3/SAMPLEPG1) EXPORT(*ALL) ACTGRP(SAMPLEPG1)
```

```
CRTPGM PGM(EASYCOMXM3/SAMPLEPG1) MODULE(EASYCOMXM3/SAMPLEPG1)
```

The generated PCML file has been modified by adding the “path” attribute into the “program” tag:

```
<pcml version="6.0">
  <!-- RPG module: SAMPLEPG1 -->
  <!-- created: 2020-03-16-10.14.34 -->
  <!-- source: EASYCOMXM3/QRPGLESRC(SAMPLEPG1) -->
  <program name="SAMPLEPG1"
    path="/QSYS.Lib/EASYCOMXM3.Lib/SAMPLEPG1.pgm">
    <data name="PARM1" type="char" length="10" usage="inputoutput" />
    <data name="PARM2" type="char" length="20" usage="inputoutput" />
  </program>
</pcml>
```

There is 2 ways to call SAMPLEPG1 program:

#### 1) Use RPG description:

```
EacXML oXml = new EacXML(ConnectionIBM);

oXml.XmlBindSrv("SAMPLEPG1");

oXml.XmlLoadDefinition("RPG", "EASYCOMXM3/QRPGLESRC,SAMPLEPG1");

string result = oXml.XmlCallProgram("SAMPLEPG1", "<PARM1>Aura</PARM1>");

XElement root = XElement.Parse(result);
//< ParameterList >
//< PARM1 Type = "Char" > Aura </ PARM1 >
//< PARM2 Type = "Char" > Aura </ PARM2 >
//</ ParameterList >

string displayResult = "";
foreach (XElement el in root.Descendants())
{
    displayResult = displayResult + el.Name + " = " + el.Value + " - ";
}
MessageBox.Show(displayResult);
```

#### 2) Use PCML on IFS:

```
EacXML oXml = new EacXML(ConnectionIBM);
oXml.XmlLoadDefinition("PCML", "/tmp/SAMPLEPG1_easycomxm3.pcml");
string result = oXml.XmlCallProgram("SAMPLEPG1", "<PARM1>Aura</PARM1>");

XElement root = XElement.Parse(result);

string displayResult = "";
```

```
foreach (XElement el in root.Descendants())
{
    displayResult = displayResult + el.Name + " = " + el.Value + " - ";
}
MessageBox.Show(displayResult);
```

## EasycomException

### *EasycomException Class*

The EasycomException class will be created by other classes methods when an Easycom error occurs. This can be trivial error like modifying a field value when no current record is locked, or any failure message coming from the AS/400.

### *Message Property*

**Class** EasycomException

#### Syntax

```
public String property Message
```

#### Description

This property is the default message, which will be shown to the user if the exception is not caught by the program.

This property usually contains the error codes, the message and full message.

### *NativeCategory Property*

**Class** EasycomException

#### Syntax

```
public EasycomNativeCategory property NativeCategory
```

#### Description

This property represents the category when having an Easycom kind exception.

The possible values are:

- **Error**: general native error. This error is generally generated locally. This can be field name invalid, ...
- **Signal**: AS/400 classic error, like file not found, ... With this category, the NativeMessage will contain the message AS/400 ID (CPF5715 for example) and NativeFullMessage the corresponding error message (File XX not found in \*LIBL).
- **Sql**: all Sql-level errors.
- **Tcpip**: all TCP/IP errors.
- **Internal**: Easycom internal error. This can be a licensing problem (see the message field for more details).



- **InternalInterface**: Internal to the PC-side interface (the .NET core). This error category can denote a bug inside Easycom or an unexpected way of using the assembly.
- **NoError**: no error was generated. This is the initial value if the EasycomException created with the empty constructor.
- **Unknown**

### *ExceptKind Property*

**Class** EasycomException

#### **Syntax**

```
public EasycomExceptionKind property ExceptKind
```

#### **Description**

This property represents the main exception level. Two values are possible:

- Easycom: the exception was caught on the Easycom level (internal in Easycom, AS/400 message, ...)
- EasycomDotNet: the exception was caught inside the Easycom .NET assembly itself (wrong field number, ...)

### *EasycomNativeErrorCode Property*

#### **Description**

Represents the native error code. The value depends on the error kind and category. If the category is Sql this is the sql error code. If the category is TCP/IP, this is the TCP/IP error code. If the category is Signal, this is the AS/400 return code.

### *EasycomNativeFullMessage Property*

#### **Description**

This property is the full native message, if available. This message comes from the AS/400 messaging system if the exception was caused by an AS/400 message ('file XX not found in \*LIBL for example), or can be an Easycom internal message if this is a logic problem (like "Invalid FieldName")

### *EasycomNativeMessage Property*

**Class** EasycomException

#### **Syntax**

```
public String property EasycomNativeMessage
```

#### **Description**



This property is a short native message, if available. This message comes from the AS/400 messaging system if the exception was caused by an AS/400 message or can be an Easycom internal message if this is a logic problem (like "Invalid FieldName").

For example, if an AS/400 message is caught, the EasycomNativeMessage will contain the message ID (like CPF5715).

NB: This value is sometime the same as EasycomNativeFullMessage.

### *EasycomNativeReturnCode Property*

**Class** EasycomException

#### **Syntax**

```
public Integer property EasycomNativeReturnCode
```

#### **Description**

Represents the native return code. This code is an Easycom-specific code.

The value can be one of the following:

2	Too many opened files
3	Not enough memory
4	Invalid pointer or Handle
5	File not found
6	Wrong key field number
10	File not allowed to be changed
13	Record locked
17	Operation sequence not valid
20	No current record
21	Null operation not applicable (field does not support null value)
24	Insufficient user rights
25	Invalid type provided
26	Info request is invalid
28	The record has changed before update
29	Already in a transaction (nested transactions are not supported)
30	Not in a transaction
22	AS/400 session ID not valid
16	Not connected
23	Wrong user or password
27	Bad property ID
12	The requested record was not found
8	Bad key length

11	The file was opened in a mode that disallows a file lock, or you have insufficient rights to perform operation
14	Begin or end of file has been reached
15	Read beyond limits of the file
6	Failed to find the field in the logical file
7	The field number is invalid

## EASYCOM Server

### Overview

Easycom server is a Software engine running on System I – AS/400.

It is compliant with all the Easycom connectors and drivers for many development tools:

WinDev & WebDev	PHP
Delphi	.NET
OLE DB	

This is the core of Easycom technology.

Basically, when installed, configured, and running, Easycom is a TCP/IP service running in a subsystem.

It is listening on a TCP Port, waiting for client connections.

Easycom Client modules are running on Windows, Windows Mobile, PASE, AIX, Linux, and many other platforms.

Easycom technology is owned by Aura Equipements company, France.

### Installing EASYCOM

EASYCOM is a Client/Server middleware. It is made of :

- A **Server engine** to be installed on System I – AS/400
- **Client connectors and drivers** to be installed on Windows, Linux or Unix workstations and servers.

### System Requirements

#### Server

- All AS/400 series B and further
- All OS/400 version from V5R2 to V7R5.
- TCP/IP protocol



### *Client*

- Protocol TCP/IP
- Operating system: all Windows OS supported by Microsoft

QSECOFR profile is required to install server on AS/400.

### **Installing EASYCOM server**

#### *Principe*

EASYCOM Server installation procedure is launched from a Windows workstation, connected to the AS/400 via TCP/IP.  
It uses FTP to upload objects on the system.

The Easycom Server installation procedure is a Windows executable file. It is embedded in the Easycom connectors installations, and automatically launched the first time you install a client connector on a Windows workstation or server.

Server has to be installed only once. If you run an Easycom connector installation again on a Windows station, you need to uncheck "Install AS/400 server" option or leave the installation procedure when the Server installation wizard is shown.

EASYCOM server consists in a set of objects (programs, commands, and files) collected into one single library, named '**EASYCOM**' (default).

It is possible to change this default library name or to [install multiple EASYCOM servers](#). In the following, library name will be referred to as EASYCOM.

#### *Prerequisites - TCP/IP and FTP*

TCP/IP must be installed, configured, and running on the AS/400 (see the CFGTCP and STRTCP AS/400 commands for more details).

**FTP is required for Easycom installation process**, but not for the EASYCOM normal operation.

The AS/400 FTP service can be started if needed using `STRTCPSVR`  
`SERVER (*FTP)` command.

#### *Confirm Destination Library Name*

Default name is EASYCOM.

We suggest keeping the default name as it is, unless you have to [install multiple Easycom servers](#) on the same machine, or you want to test a new version without updating the existing one.

The library will be created if it doesn't already exist.

If the library already exists, a backup copy will be created in library EAC\_BACKUP.

In the future, you need to rename this server library, or copy it, you will need to run [CFGEACTCP](#) command, using the new library name, to link the objects together in the new library.

### *Installing the demonstration files*

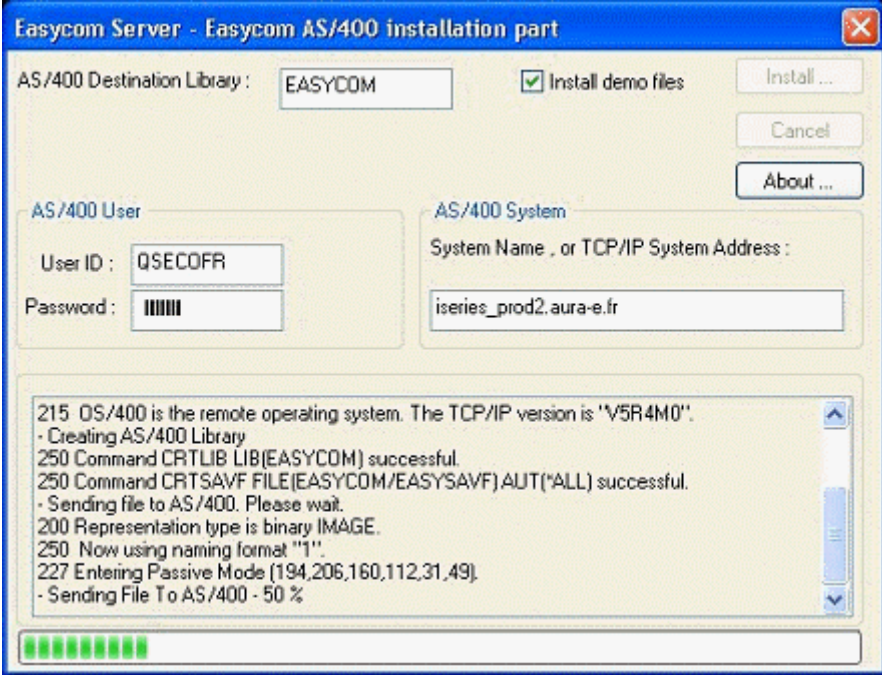
For the first EASYCOM installation on AS/400, the demonstration files allow to run the test and demonstration programs installed on the client workstation within development environment.

### *Profile for installation*

**QSECOFR profile is required for proper installation.**

**Give the AS/400 name or the AS/400 IP address on which the software will be installed.**

**Enter a username and password to proceed with the installation.**



It is not recommended to use any other user than QSECOFR.  
Some EASYCOM library objects are configured to be owned by QSECOFR.  
The EASYCOMD (\*PGM) object must be run under QSECOFR permissions.

If QSECOFR is not used for installing the server, the auto-configuration may not be completed, and the first start-ups may be difficult.

### *Installation result*

#### EASYCOM Subsystem

When installation is completed, **Easycom subsystem is started**.

This subsystem must remain active to accept client connections. See your system administrator to have the [subsystem started at IPL](#).

#### Operations performed on AS/400

Creation of an EASYCOM library and restoration of some objects in this library. [CFGECTCP](#) and [EACINSTALL](#) commands are automatically run by the installation process.

#### Operations performed on PC

Creation of an \EASYCOM folder and specific subfolders and copy of various files.

### *Installing an additional EASYCOM server*

An Easycom Server has the following properties:

- A library with all the objects (Default name = EASYCOM)
- A Subsystem (Default name = EASYCOM)
- A TCP Port (Default = 6077)

To setup an additional server on a System, you need to install EASYCOM in a new library. The subsystem name must be unique, and a new unique port number must be assigned.

Proceed with the installation of Easycom server, from a Windows workstation.

Give a new unique name to the library in the installation wizard (Example EASYCOM2).

Once the library is installed, you need to create the new subsystem and assign a port number, by running command CFGECTCP.

Example: To install an additional Easycom server, in library EASYCOM2, subsystem EASYCOM2, port 6078, run the following commands:

```
ADDLIB EASYCOM2
```

```
CFGECTCP LIB(EASYCOM2) SBS(EASYCOM2) PORT(6078)
```

On the client workstations, you need to configure Easycom client, or applications, to connect to the right Easycom server.

Add the port number at the end of the name or address of the AS/400 to connect to, separated by a colon (:).

Example:

```
SYSTEMAS:6078
```

```
192.168.0.10:6078
```

You need to change this value with "Easycom configuration" utility, if the system is the default one, or in the connection properties of your application.

## EASYCOM behavior

### EASYCOM server configuration

The AS/400 EASYCOM library contains all EASYCOM objects.

When the server is launched the first time a single additional object is created: EASYCOM object, \*FILE type in QGPL library.

An EASYCOM library entry is automatically added to the online libraries list (ADDLIBLE) for each job started on client workstations. Therefore, no explicit addition to the 'users' JOBD is required.

The following objects may be modified in the EASYCOM library:

AURA	*FILE	Modified when the user license is registered.
EACSESSION	*FILE	Modified when the user license is registered.
YPROCHDR	*FILE	Modified by adding native AS/400 program description to be called by client applications. This file can be moved to another library listed in the jobs LIBL.
YPROCPARMS	*FILE	Modified when AS/400 native programs calling parameters are described. This file can be moved to another library listed in the jobs LIBL.
CFGEAC	*DTAARA	Holds parameters setup by CFGEAC command
CFGEACSSO	*DTAARA	Holds parameters setup by CFGEACSSO command
EASYCOM	*SBSD	Updated when CFGEACTCP is used
EAC_EIM	*USRSPC	Holds parameters setup by CFGEACSSO command in *EIM mode.

If the AS/400 EASYCOM server is updated, these objects are eventually upgraded to new format, but the settings are kept except for EASYCOM subsystem.

### IPv6 connectivity

Easycom is fully compatible with IPv6 networking.

There is no special configuration to do to allow IPv6 connections. The only requirement is using a minimum version for EASYCOMD, EASYCOM programs, and the OS/400 version:

- EASYCOM program must be 4.60.10 or above.
- EASYCOMD program must be 3.0.3 or above.
- OS/400 version must be V5R3 or above.

To have full IPv6 connection the Easycom client must also be compatible with IPv6. You need to check the documentation for each client products.

This is recommended to use a name instead of an IP address. However, this is possible to specify an IP address in IPv6 syntax, like follows:

- 2001:db8::1428:57ab

- or [2001:db8::1428:57ab] :6077 with 6077 as a port number.

You can check whenever the connection was made in IPV6 or not using the NETSTAT command of the OS/400.

You also can see it in the log file (generated by CFGEAC command or by the client application).

#### Notes :

- The IP address will appear in the IPv6 syntax in all exit programs.
- The IP version is available in two new exit programs: EACTCPP01 and EACLOG002.
- Connections in V6 and V4 are both accepted by default. You can use EACLOG002 to deny connections if needed.

### Pre-start jobs

"Pre-start" jobs, anticipate job starting, and speedup EASYCOM connection. Its use is particularly well suited if applications are frequently connected and disconnected (i.e., Web applications).

#### **Advantages:**

- Faster connection Start (quasi-instantaneous connection).
- Possible to initialize a custom environment before the connection is started (however the username is not known at this stage).

#### **Disadvantages:**

- The jobname is equal to the name that was chosen in the ADDPJE command, cannot be changed during the connection.
- The effective user is not visible in WRKACTJOB displayed list except on V5R4 and above. Effective user can be known with WRKACTJOB option 5, and option 1 (job status).

Here are the required commands to setup prestart jobs for Easycom (this will end all active connections):

**ENDSBS SBS(EASYCOM) OPTION(\*IMMED)**

**ADDPJE SBS(D(EASYCOM/EASYCOM) PGM(EASYCOM/EASYCOM) INLJOBS(10)  
JOB(EASYCOMPJ) JOBD(EASYCOM/EACJOB) CLS(EASYCOM/EACCLS)**

**CFGEACTCP LIB(EASYCOM) PJ(\*ON)**

To return to backward configuration, stop the subsystem, remove PJ definition using RMVPJE, and then run again CFGEACTCP set to \*OFF.

### IPL process

EASYCOM subsystem must be active to accept client connections.  
EASYCOMD job must be active in the subsystem.

See your system administrator to start EASYCOM subsystem during the IPL of your system. One way to have EASYCOM started at IPL, is by changing QSTRUP program in QSYS library.



Retrieve the source of this program:  
RTVCLSRC QSYS/QSTRUP ...

Change the source, by adding command "STRSBS EASYCOM/EASYCOM" after TCP/IP is started.

And compile:  
CRTCLPGM ...

TCP/IP must be on when Easycom subsystem is started!  
Starting Easycom subsystem, automatically starts EASYCOMD job.

If EASYCOMD is not started, check EASYCOMD \*JOB, and see message queue EACMSGQ in Easycom library:  
DSPMSG EACMSGQ

### Default LIBL

Easycom Client job is started with an initial library list in the following order:

- Libraries from System library list.
- Libraries resulting from EACJOB job description, if it exists.
- Libraries resulting from the Job Description assigned to the user.
- Easycom library if it is not already in one of the above lists.

Default LIBL for EACJOB job description is \*NONE, as defined by command CFGEACTCP.

**Warning!** The initial LIBL was changed with EASYCOM Server **version 4.58.80**

Client application can change the initial LIBL for its Easycom job.  
It uses remote command function API to run ADDLIB, RMVLIB or CHGLIB.

### Default CCSID, SRTSEQ

Easycom is using the default CCSID of the system.

It is possible to give other values of use the values of the user profile using the

### Timeout on call program

Using CFGEAC program allows configuring a maximum execution time (timeout) for a program call.

When the program execution is lasting above configured time, the program call is simply cancelled, and an error is reported to the client.

Warning: when the timeout condition is met, some files and/or resources allocated by the called program may not be released, and never will, until the Easycom job is ended.

### Protecting access to EASYCOM

EASYCOM takes care of security at user level: an EASYCOM program can be used on an AS/400 after username and password validation on the system. The processes carried out by the program will be executed under the user identity. All AS/400 permissions of that user will be applied.

All AS/400 rights will be applied.



Each connection opened requires a valid profile and password, or a Kerberos ticket if this kind of connection is allowed and configured.

Advanced security settings is available using the following Easycom Exit Programs:

- [EACP003](#) : authorizes a program according to a complementary password (independent of the profile). This allows locking the easycom server for only a set of applications and/or developers.
- [EACTCP003](#) and [EACTCP002](#) : controls the client settings like IP address, changes effective user or performs specific job submission.
- [EACTCPP01](#): controls security just before validating the login (avoids login exchange if the client is denied from this IP address or protocol).
- [EACLOG002](#): controls security just after the login has been validated by Easycom.
- [EACSSO001](#) : controls Easycom Single Sign-On system.

### Single sign on - EIM

#### *What is EIM ?*

The Single Sign On (SSO) in the EIM mode is the implementation of the IBM Single sign-on system.

The main idea is that there is one unique credential management server, Kerberos. When the user is connecting to its station the Kerberos server gives him a **ticket**. When EIM is used during connection that **ticket** is used in place of user/password. This ticket is validated by the Kerberos server (from the iSeries job), and a corresponding OS/400 user is given from the ticket username (the windows login).

So, the user password is not used anymore, and best security is to put "\*\*NONE" to the password. This way the user **must** use a Kerberos authentication to connect to the system.

#### *EIM installation on IBM i*

EIM Installation on AS/400 consists of the following steps:

- Create a domain in EIM.
- Add the domain in the domain management.
- Create a source user registry definition in EIM.
- Create a user identifier in EIM.
- Create a target association in EIM for the user identifier.
- Create a source association in EIM for the user identifier.
- Test the connection to the EIM domain controller.
- Configure the EIM Identity Token Connection Factory.

All required information can be found here:

<https://www.ibm.com/docs/en/was/9.0.5?topic=eim-configuring>

Once it works with Client Access you can setup Easycom.

### *EIM with Easycom*

To use EIM with Easycom we need to do the following:

1. Install and configure it in the AS/400 and the domain controller.
2. Grant the TCP user to access the keytab file. QTCP is the user for EASYCOMD job.

```
CHGAUT  
OBJ('/QIBM/UserData/OS400/NetworkAuthentication/keytab/krb5.key  
tab') USER(QTCP) DTAAUT(*R)
```

3. Enable the Kerberos authentication:

```
CFGEACAUTH LIB(EASYCOM) KERBAUTH(*ON)
```

Note: Instead of Kerberos authentication you also can use client certificate authentication, with certificates registered in the EIM database.

4. Configure Easycom to use EIM on the server,  

```
CFGEACEIM LIB(EASYCOM) ACTIVE(*YES) EIM_LDAPU(administrator)  
EIM_LDAPPW(xxx)
```
5. Optionally define an exit program [EACLOG002](#)
6. Update applications to use EIM by using \*KERBAUTH special value for the login.

EIM implementation on client is very simple. All you need is to specify "\*KERBAUTH" special value for the user id, and a recent client DLL. The password has no importance (can be blank or any value).

There are special TCP/IP error codes (negative) for different Kerberos errors (ticket expired, ...), with corresponding native error text (coming from i5 or from client).

For testing you can type \*KERBAUTH in place of the username and leave a blank password. After this, you can put that special value in your client/server programs.

### *EIM common problems*

#### **Domain names must match.**

The domain name that is configured with iSeries navigator must match the domain name of the machine.

If not, you will get an error on the client like: *"the specified target is not known or inaccessible"* (with tcp/ip error code -14)

Here is how to check it:

Step 1: to know what the real domain name is, do the following using a command prompt on the **client machine**:

Enter "nslookup", then type the name of the iseries, like follows:

```
Default server : domain_controller.domain-name.com
Address: 194.206.160.4

> my_iseries
Server : domain_controller.domain-name.com
Address: 194.206.160.4

Name : my_iseries.domain-name.com
Address: 194.206.160.112
```

So here the correct domain name is **domain-name.com**

Step 2: check that exported keytab contains the correct domain name.

Do do this, use iseries navigator, and go to "security", and then "Network authentication service". Right-click and select "Manage keytab". Click on the "Details" button.

You should see a line with:

Principal Type: i5/OS

Principal Name: krbsvr400/my\_iseries.domain-name.com@DOMAIN-NAME.COM

Where DOMAIN-NAME.COM is your i5/OS realm.

If this is not correct, you need to modify configuration a re-export keytab, or you need to check your DNS to have matching domain names.

### DES encryption must be enabled on the DC accounts created from keytab.

If not, you will get an error "*Encryption or checksum type is not supported.*"

To enable it, you need to connect to the domain controller machine, and run the Active Directory application. Then, select "Users", and choose a user named:

my\_iseries\_1\_krbsvr400

(There also can be others: my\_iseries\_2\_krbsvr400, ...)

On the properties of that user, choose "Account", and check "use DES encryption".

### Error on connect: "Not authorized to access key table".

The keytab file must be accessible from the i5/OS account that is used for EASYCOMD, typically QTCP.

You need to know the location of the keytab file. iseries navigator, and go to "security", and then "Network authentication service". Right-click and select "Manage keytab". Follow the wizard until the last step (you can cancel it if you already done the wizard). The keytab file path is specified in that window.

The typical location is:

/QIBM/UserData/OS400/NetworkAuthentication/keytab/krb5.keytab

To grant access to QTCP you need to do the following command:

CHGAUT

OBJ('/QIBM/UserData/OS400/NetworkAuthentication/keytab/krb5.keytab')

USER(QTCP) DTAAUT(\*R)

### The time of all machines must be synchronized.

If you get errors like '*ticket not yet valid*' or '*ticket is expired*', this is probably due to wrong time synchronization.

Check QTIMZON and QTIME system values using WRKSYSVAL. Also check the time clock and time zone for the domain controller and end-users' machines.



## SSL

### SSL connection - prerequisites

Easycom connection can use SSL encryption.

The main prerequisites for using this feature are:

- EASYCOM version must be 4.60.10 or above.
- EASYCOMD version must be 3.0.3 or above.
- OS/400 version must be V5R3 or above, with i5/OS Host Servers (57xx-SS1 Option 12), Qshell Interpreter (57xx-SS1 Option 30).
- An application ID named 'EASYCOM' must be created in the OS/400, using DCM. A certificate must be assigned to the application.
- System i™ Access for Windows® (57xx-XE1).
- The Easycom server must allow SSL connections using CFGEAC.
- The client must support SSL and have the certificate of authority (CA) installed (the CA from which is issued the certificate assigned to the 'Easycom' application).

SSL client support depends on the product versions and on the platforms that are used. You need to check the documentation of the client products.

### SSL connection - client configuration

To enable Windows SSL you need to add the certificate to easycom.ini or Easycom configuration.

Be careful, there is a difference between Windows SSL and AS/400 SSL.

Easycom using by Windows OS use Windows SSL, and you have to configure it to use it.

### SSL connection - server configuration

To enable SSL in Easycom you need to create an **application** and assign a **certificate to it**. The application ID must be equal to Easycom. The certificate must have been issued by a CA that will be accepted by the client.

To create the application, you will need to use the **Digital Certificate Manager (DCM)** of the AS/400.

Exactly the same configuration is required to enable SSL connection with Telnet (apart for client part).

Here are the required steps for the server configuration:

- First, connect to the DCM using a web browser, with **http://my\_iseries:2001** and then click on "Digital Certificate manager" (a tip says that it is for creating and managing digital certificates).

If this doesn't work, you will need to enable it using iSeries navigator.

- Then, click on "**Select a Certificate Store**", and select "**\*\*SYSTEM**", then click "**continue**". This will prompt you to enter the password for the certificate store.
- Then select "**manage applications**" on the left menu and click on "**Add application**". Then select "**Server**", and click "**continue**".

Enter "**EASYCOM**" for the application ID. This is the key that will be used by Easycom. Enter a description and validate.

- Now we need to **assign a certificate to the application**. This is a required step: the certificate is used to ensure that the server can be trusted and for encryption. There are two options for that :
  1. You can generate the certificate using the AS/400 CA (Certificate of Authority). In this case the CA certificate will need to be installed on the client (first, export the CA certificate using the export menu).
  2. You can request a certificate from a trust 3<sup>rd</sup> party CA. In this case you will need to import it into the \*SYSTEM certificate store using the "import" menu.

To assign the certificate, click on "**Manage Application**", and then "**Update certificate assignment**". Choose "**Server**" and click "**continue**". You will see the current assignment ("*none assigned*") for the application.

Select the 'Easycom' entry that you have created and click on '**Update Certificate Assignment**'. Select the appropriate certificate and click on 'Assign New Certificate'.

Now click on "**Validate**": this will check that the certificate is valid for the system.

- 
- Finally, configure Easycom server to use SSL using [CFGEAC](#):

```
CHGCURLIB EASYCOM
CFGEAC LIB(EASYCOM) SSL(*ON)
```

- Then you need to restart EASYCOMD with the following command:

```
STREACD PORT(*JOBID) RESTART(*YES)
```

- Then try a connection from a client using SSL. You can use the [Easycom Configuration](#) tool for that.
- You can check the options using the following command:

```
DSPMSG EASYCOM/EACMSGQ
```

This will show:

```
EASYCOMD:Starting from library EASYCOM, Version 3.00.03, (Nov 10 2008
11:15:49/OS530).
EASYCOMD:EASYCOM - (c)AURA Equipments - http://www.easycom-aura.com
----- Lib=
;Pwd=SSL support
EASYCOMD:Configuration used for Library EASYCOM is Dq=
SSL=On
```

In case of problem, the errors will appear here. Note: this does not ensure that the connection is in SSL, but only that SSL will be accepted.

To know if SSL is used during a connection, use [EACLOG002](#) exit program. You usually also can check it in the client application.

[Easycom Configuration](#) tool is showing SSL status on the connection test page.

To check it for an active job, look at the call stack of the job. To do this, use WRKACTJOB command, then option 5, and then option 11. If you see "SSL\_Read" in the stack, this means that the connection is using SSL.

### SSL connection - client certificate

Easycom can accept client certificates for two purposes:

- Additional security of the network. The server can give access only to clients that have a valid certificate.



- Use the client certificate to assign the OS/400 user to use. The client certificate subject can be used to define the OS/400 username, or the EIM database can be used for this.

The client certificate must be valid for the AS/400. The certificate is considered valid if it is issued by one of the CA (Certificate Authority) that are installed on the AS/400, in the \*SYSTEM certificate store.

So, the certificate can be issued by the AS/400; in this case the CA is the Local CA.

### *Create a X.509 registry in EIM, and configure LDAP location (optional)*

This step is required if you want to use the EIM database to map the certificate to the OS/400 user.

In this case the supplied username must be "\*SSL".

Using system i access, go to "Network"/"Enterprise Identity Mapping"/"Domain Management"/"<your domain>"/"User Registries", and click "Add a new system registry".

Choose a name, and "X.509" registry type.

Under "configuration", select properties, and select the X.509 registry just created.

Now we need configuring the LDAP location for the \*SYSTEM store. This will make the user certificates creation process linked to the EIM.

Use Digital Certificate manager. Connection is at: [http://my\\_iseries:2001](http://my_iseries:2001). Select "Digital Certificate Manager" (on V6R1 select "i5/OS management" and then "Internet configuration" first. Logon as QSECOFR when prompted).

Select "Manage LDAP location", and enter:

LDAP server: fully defined host name: my\_series.mydomain.com

Directory distinguished name (DN): dc=

Use Secure Sockets Layer (SSL): No

Port Number: 389

Login distinguished name (DN): cn=

Password: xxxx (password for LDAP used by EIM).

### *Create a user certificate*

Go to [https://my\\_iseries:2010/QIBM/ICSS/Cert/Admin/qycucm1.ndm/main0](https://my_iseries:2010/QIBM/ICSS/Cert/Admin/qycucm1.ndm/main0) using the user login for which you want to create the certificate.

Then select "Create Certificate". The login name will be the user under you connected to the web site.

Then click on "install certificate". This will install the certificate into the web browser. Then you can export it into a portable format if needed.

If you created the X.509 registry and specified the LDAP location the DCM configuration, the EIM settings is automatically updated. Note: an EIM mapping MUST exist for this user before doing this (with an i5/OS target equal to that user).

### *Install the user certificate on your local store*

Use the web browser to transfer the user certificate locally.

### *Enable the Easycom server part*

CHGCURLIB EASYCOM

[CFGEACAUTH](#) LIB(EASYCOM) SSL(\*ON) SSLAUTH(\*ON) SSLROLE(\*EIM)



Use "SSLROLE(\*EIM)" if you use a X.509 registry or \*SUBJECT if you use the certificate Distinguish name for username.

EIM must be configured with CFGEACEIM as well.

You can try connections with "\*\*SSL" userprofile and no password if EIM is activated, or with a regular user and password if not.

Now type DSPMSG EASYCOM/EACMSGQ. You should see:

```
EASYCOMD:Starting from library EASYCOM, Version 3.00.05, (Jun 23 2009 16:29:38/OS530).
EASYCOMD:Eim connection OK - X.509 registry is 'p520 certificates'
EASYCOMD:EASYCOM - (c)AURA Equipments -
http://www.easycom-aura.com
=====
EASYCOMD-V.3.00.05 (EASYCOM/EASYCOMD); Lib=EASYCOM; PJ=Off; SSO=Off;
Eim=On; Pwd=2; Port=6077; IPv6; SSL
EASYCOMD:Configuration used for Library EASYCOM is Dq=EASYCOM, Vers=
KerbAuth=Off, SSL=On, SSLAuth=On *EIM
```

This shows the the X.509 (certificates) registry is detected and named 'p520 certificates'.

This also confirms SSL capability for EASYCOMD.

This also shows (from first connection attempt) that the EASYCOM library is with SSL activated, and SSL authentication activated with \*EIM role.

If there is a problem with authentication a message will appear here.

## EASYCOM jobs on AS/400

### The jobs on AS/400

When a client application connects to Easycom Server on System I – AS/400, an Easycom Client job is submitted in Easycom Subsystem.

This job run under the authority of the connected user. It can "adopt the authority" of another user on request of the client application.

If exit program EACTCP003 exists in Easycom library, it can submit the client job according to its own rules and descriptions.

If job description EACJOB exists in Easycom library, the job is submitted according to it. Otherwise, it is submitted according to the Users job description.

In any case, user initial library list will be added to the Easycom Client job.

Priority of Easycom Client job is defined by class object EACCLS in Easycom library. This priority can be adjusted with CHGCLS command.

System can use [prestarts jobs](#).

### Jobs creation and properties

The job alternatively can be created by the safety program EACTCP003 (see below),

If EACTCP003 does not exist or the job does not start, it is created according to:

- EACJOB, if it exists,
- The JOB associated to the user profile that is authenticated, if EACJOB is not present.
- The JOB associated to the user profile that is authenticated for the LIBL management (see [Default LIBL](#))



EASYCOM on AS/400 works using a subsystem and a daemon. That daemon handles the connection requests from client applications. When the application is launched, and a connection established with AS/400, a job is created on AS/400. There is an active job for each connected client application, using the appropriate authority and user rights. Each application can have its own file openings, locks, current positions, and transactions in progress.

### EASYCOM job priority

Jobs are stored in EASYCOM subsystem. It uses **EACJOB** for its description and **EACCLS** for its priority class. Subsystem priority class can be modified with CHGCLS command.

### EASYCOMD authority

EASYCOMD (\*PGM) is submitted in Easycom subsystem according to EASYCOMD (\*JOB) Job description.

EASYCOMD program has special authorities. Those authorities are necessary to handle security features and submit jobs (or work with prestart jobs) for other users.

To have those features EASYCOMD program is owned by QSECOFR, as '\*OWNER' user profile and is using 'adopt authority'. By default, EASYCOMD job is submitted under QTCP but using QSECOFR user rights because of those properties.

If EASYCOMD has wrong properties, you can restore them with the following commands:

```
CHGPGM PGM(EASYCOM/EASYCOMD) USRPRF(*OWNER) USEADPAUT(*YES)
CHGOBJOWN OBJ(EASYCOM/EASYCOMD) OBJTYPE(*PGM) NEWOWN(QSECOFR)
GRTOBJAUT OBJ(EASYCOM/EASYCOMD) OBJTYPE(*PGM) USER(QTCP) AUT(*USE)
```

## EASYCOM Server configuration commands

### CFGEAC (Configure Easycom)

**CFGEAC** command allows configuring EASYCOM server properties on iSeries - AS/400 system.

```

EASYCOM SERVER CONFIGURATION (CFGEAC)
Type choices, press Enter
Easycom server library name . . . > EASYCOM          Alpha value
Easycom job priority . . . . . *DFT                  1-99, *SAME, *DFT
TCP/IP Keep Alive frequency . . 120                  seconds
Delay before asking again pwd . *NONE                seconds
Delay before automatic SIGNOFF  *NONE                seconds
Easycom Log File level . . . . . *NONE              Number, *SAME, *NONE
Print the clock in Log File . . *NO               *SAME, *YES, *NO
Automatic Keep Alive start . . . *YES              Number, *YES, *NO, *SAME
Detailed Job Log . . . . . *NO                *SAME, *YES, *NO
Lock Easycom host . . . . . *NO                *SAME, *YES, *NO
Time Out on Ext Pgm Call . . . . *NONE          Number, *SAME, *NONE
Character Set ID . . . . . *USRPRF              -2-
65535, *USRPRF, *SYSVAL...
Sort sequence table . . . . . *NONE            Name, *USRPRF, *SYSVAL...
Library . . . . .                               Name, *LIBL
Convert CONCAT field to A type  *NO              *SAME, *YES, *NO
SSL enable . . . . . *OFF                     *SAME, *OFF, *ON, *ONLY
End
F3=Exit F4= F5=efresh F12=Cancel F13=How to use this display
F24=More keys
```

### EASYCOM server library name (LIB)

Enter the library name in which the EASYCOM server is installed.

**EASYCOM server job priority (PTY)**

This parameter is used to override JOBBD job priority setting. If set to 0, JOBBD determines job priority. JOBBD used with EASYCOM is EACJOBBD.

**TCP/IP Keep Alive frequency (TCPTOUT)**

This parameter is used to set the 'keep alive' interval value. Default value is 120 seconds. When 'keep alive' is on, a TCP/IP message is sent from PC to AS/400 every n seconds. This is useful to keep a remote line up, and to have automatic shutdown of jobs that are no longer linked to a client application, even in case of client crash.

If the AS/400 EASYCOM job does not receive the message in n+10 seconds, it automatically shutdowns.

These TCP/IP messages are only sent when the communication is idle for that delay.

This value can be set to 0 to disable it. This is useful when debugging, as some debuggers avoids Easycom to send the TCP/IP message when the process is stopped.

This parameter can also be set using 'Easycom configuration' tool on the PC.

**Delay before asking again pwd (RESIGN)**

This parameter is used to make end-user sign-on again after a given idle time. (Default is disabled).

*Not currently supported.*

**Delay before automatic SIGNOFF (CONNECTION)**

This parameter is used to close a connection after a given idle time.

**EASYCOM Log File level (LOGLEV)**

Use this to enable an AS/400 log file. Valid values are 1 to 4.

It will create a EASYCOM/LOGFILE(MEMBER) file, EASYCOM is the Easycom installation library, MEMBER is Easycom job name.

Be careful with that:

- Log file member is always cleared when a new connection is made
- If two jobs with same job's name are run, the second cannot have log file and will be locked for 1 minute at start-up.

**Print clock in Log File (LOGCLOCK)**

Allows getting time information in log file: command processing starting and ending time, CPU consuming.

**Automatic Keep Alive start (HBEAT)**

If this value is \*YES, the 'keep alive' message (see above) will be generated unless the PC is configured to refuse it. If this value is \*NO, 'keep alive' will not start unless the PC is configured to enable it.

**Detailed Job Log (JOBLOG)**

This option is used to run an automatic job login. This can be changed with Easycom JOBBD (EACJOBBD).

**Lock EASYCOM host (LOCKED)**

Easycom is default locked if this option is used. This means that the Easycom connection is accepted, but no file neither program access will work until the 'unlock' password arrives. See our documentation about EACP003 entry program for more information.

**Time Out on Ext Pgm Call (PGMTOUT)**

Defines a timeout for program execution. This avoids program call taking too much time. When the timeout is reached, the call will abort and Easycom will return an error.

**Character Set ID (CCSID)**

Indicates character set used with EASYCOM. Default character set is \*HEX (65535).

A good idea can be to set it up to \*USRPRF.

**Sort Sequence table (SRTSEQ)**

Indicates the sort file to be used for comparison and sorting. Possible values are those suitable for SRTSEQ parameter in the system CHGJOB command. \*LANGIDUNQ is a value that allows "natural" sorting for the current CCSID. However, there is a need to be careful to have the indexes or logical files with a compatible sort sequence.

**Convert fields CONCAT to type A (CONCATF)**

Indicates if CONCAT operations resulting fields must always be considered as alphanumeric type fields. Possible values are:

\*YES : CONCAT result fields will be processed as a single alphanumeric type field.

\*NO : CONCAT result fields keep their original type.

**SSL enable (SSL)**

Specifies how SSL encryption can be used with Easycom.

Use DSPMSG EACMSGQ to know if SSL init worked.

Note: Modifying this option requires EASYCOMD job restart. You can perform it using STRSBS/ENDSBS system commands or the STREACD command.

Possible values are:

\*YES: Both SSL encrypted, and clear connections are accepted.

\*NO: SSL is not used on this library. SSL connection attempts will fail.

\*ONLY: SSL usage is mandatory. The connection will fail if the client does not support SSL, or if SSL negotiation failed.

**CFGEACTCP (Configure Easycom TCP/IP)**

This command is automatically called when automatic installation is performed.

It creates the subsystem and all the related objects, and it sets the TCP/IP port number used by Easycom service.

Objects created: SBSDB, JOBD, JOBQ, CLS.

Object name	Object type	Description
EASYCOM (default)	*SBSDB	Subsystem in which service is running, and client jobs are submitted.
EACJOBDB	*JOBDB	Descriptions for client jobs.
EACJOBQ	*JOBQ	Client job queue.
EACCLS	*CLS	Class for client jobs.
EASYCOMD	*JOBDB	Job description for the EASYCOMD job, which must always be active in the subsystem.

Note: this command creates the required objects to have a subsystem running and starts it. But it **doesn't store any of the parameters**.

### **EASYCOM library (LIB)**

Enter EASYCOM server objects library name, where new sub-system will be created, with all related objects.

If the objects already exist in the library, they will be replaced.

### **System library in LIBL (SYSLIB)**

This parameter is no used.

### **EASYCOM sub-system name (SBS)**

Enter the subsystem name to be created in the library.

The subsystem name must be unique on the system. If you have more than one Easycom Server running on the system, each server must have its own library and subsystem. See [Installing an additional Easycom Server](#).

When the subsystem will be active, job associated with each connection will run in this subsystem.

### **EASYCOM service port (PORT)**

Enter the TCP port number to be assigned to the EASYCOM server. If multiple EASYCOM servers will run on the same machine, a different port number must be assigned to each one.

Possible values are:

**\*DFT** : If a service named easycom exists in the port services table, the associated port will be used. See WRKSRVTBLE system command to manage the services table. If Easycom service does not exist, **default port 6077** is used.

**Number** : port number to be allocated to new EASYCOM server.

If port number is changed, it must be changed in the client configuration, using "Easycom configuration utility", or by changing the connection properties in your client application.

When a non default port number is configured, port number must be added at the end of the server's name or address in the client application, separated by a colon (:).

Example: my\_server:6090

### **Authorize pre started jobs (PJ)**

Possible values are:

**\*OFF**: "Prestart" jobs are not used when client session requires a connection, even if they are configured in the subsystem.

**\*ON / \*AUTO**: To use pre-started jobs on the subsystem if they are configured and active.

This option only authorizes Easycom to use pre started jobs at connection time, if they are active.

You need to configure the pre started jobs manually (available on V4R4 and above).

To configure pre started jobs, after running CFGEACTCP command, you need to proceed as following:

#### **Stop EASYCOM subsystem.**

Example: ENSSBS EASYCOM \*IMMED

#### **Run ADDPJE command.**

Example:

```
ADDPJE SBSD (EASYCOM/EASYCOM)
PGM (EASYCOM/EASYCOM)
USER (QUSER) INLJOBS (4)
JOB (PJEASYCOM) JOBD (EASYCOM/EACJOB)
CLS (EASYCOM/EACCLS)
```

**Start subsystem again:**

```
STRSBS EASYCOM/EASYCOM
```

**Comments:**

Command CFGEACTCP starts the Easycom subsystem.

TCP/IP must be on when subsystem is started!

Starting Easycom subsystem, automatically starts EASYCOMD job.

If EASYCOMD is not started, check EASYCOMD \*JOB, and see message queue EACMSGQ in Easycom library:

DSPMSG EACMSGQ

***STREACD (EASYCOM service start)***

**STREACD** command starts EASYCOM service. EASYCOMD program is started in the subsystem to allow connection of clients' stations.

Remark: this command **doesn't store any of the parameters**.

**EASYCOM Library(LIB)**

Enter EASYCOM server library name, where subsystem description was created.

**EASYCOM service port (PORT)**

Enter the TCP port number assigned to EASYCOM server. If several EASYCOM servers will run on the same machine, a different port number must be assigned to each one. Possible values are:

\*DFT: If a service named Easycom exists on the port services table, the associated will be used. See WRKSRVTBLE system command to manage the services table. If Easycom service does not exist, **default port 6077** is used.

\*JOB: The service is started according to EASYCOMD job description in the library.

Number: Port number to be allocated to new EASYCOM server.

**Authorised pre-starts jobs (Pre-starts jobs - PJ)**

This parameter is used only if PORT parameter is different from \* JOB. Use pre-starts jobs in the subsystem.

Possible values are:

\*OFF (default): "Pre-start" jobs are not used when client session requires a connection, even if they are configured in the subsystem.

\*ON / \*AUTO: To use pre-started jobs on the subsystem if those are configured and active.

**EASYCOMD restart (RESTART)**

Stop and Start again EASYCOMD job if it is already running in the subsystem. Possible values are:

\*NO: If EASYCOMD job is already active, it remains unchanged.

\*YES: If EASYCOMD is running, it is stopped, then re-started with new parameters.



EASYCOM subsystem must be active.

EASYCOMD job (demon) runs permanently in the EASYCOM subsystem. It starts automatically when subsystem is started.

EASYCOMD uses TCP/IP port 6077 (default) to accept connection requests from client stations.

If a safety system or another application prohibits using this port, it can be modified with CFGEACTCP command.

### **Comments:**

TCP/IP must be on when EASYCOMD is submitted.

If EASYCOMD is not started, check EASYCOMD \*JOB, and see message queue EACMSGQ in Easycom library:

DSPMSG EACMSGQ

### ***EACINSTALL (Easycom Install)***

This command is the final setup command. This command updates Easycom objects to have the best possible match according to the current running OS/400 release.

This command changes the default SQL interface, and EASYCOMD program to support EIM.

```
EASYCOM INSTALLATION (EACINSTALL)
Type choices, press Enter.
Easycom Library . . . . . EASYCOM Lib. of product EASYCOM
OS VERSION FOR ADJ. . . . . *AUTO      MINIMUM OS VERSION FOR ADJ
LEVEL OF SQL INTERFACE TO USE . . *AUTO      *CISC, *EMBED, *CLI, *AUTO
```

You can change the default SQL INTERFACE from \*CLI to \*EMBED. This will use the embedded SQL interface in replacement of \*CLI.

The \*CLI interface is more powerful but using \*EMBED can help solving issues that are encountered by \*CLI interface. The \*CISC interface is obsolete and is no longer included in latest versions of Easycom.

The \*EMBED interface limitations are: cannot use LOB fields, or SQL procedures. However, in some cases it is fastest than CLI.

In fact, the \*EMBED is the old - historical – interface, and \*CLI is the one. Only the \*CLI interface will have future improvements.

### ***CFGEACAUTH***

This command configures the authentication methods and security options which are valid with Easycom.

```
Easycom Authentication config (CFGEACAUTH)
Type choices, press Enter.
Easycom server library name . . > EASYCOM Alpha value
Use SSL encryption . . . . . *OFF      *SAME, *OFF, *ON, *ONLY
Use SSL authentication . . . . . *OFF      *SAME, *OFF, *ON, *ONLY
SSL authentication role . . . . *SAME
Use Kerberos authentication . . *ON      *SAME, *OFF, *ON, *ONLY
```

### **Use SSL encryption**

This option defines if the SSL encryption is supported, or mandatory. Possible values are:

\*OFF: SSL is not used by the Easycom server.

\*ON: SSL is used if requested by the client

\*ONLY: SSL must be used. The connection will be rejected if the client doesn't support SSL or if the SSL negotiation fails.

### Use SSL authentication

This option defines if SSL authentication is enabled. This option is valid if 'Use SSL encryption' is activated. Possible values are:

\*OFF: SSL authentication is not accepted.

\*ON: SSL authentication is valid. A valid certificate must be provided by the client.

\*ONLY: SSL authentication is mandatory. A valid certificate must be provided by the client. This SSL authentication can validate the OS/400 user or can only act as an additional security option (see 'SSL authentication role').

### SSL authentication role

This option defines how the SSL authentication will imply an OS/400 user. Possible values are:

\*NONE: the SSL authentication won't define an OS/400 user. The client certificate will be checked by Easycom, but not used to define the OS/400 User. OS/400 User and password, or Kerberos authentication must be provided as well.

\*EIM: Easycom will search if the client certificate is found in the EIM database. If yes, the EIM will define which user to use. In this case EIM configuration must be valid.

\*SUBJECT: the certificate subject is equal to the OS/400 username. In this case the EIM configuration is not necessary. The SSL client certificate will be used for the whole authentication process.

### Use Kerberos authentication

This option defines if the Kerberos authentication is valid. The EIM configuration must be valid to be able to map the Kerberos authentication (typically Windows credentials) to an OS/400 user.

## CFGEACEIM

This command is designed to configure the EIM connection for Easycom. It replaces the CFGEACSSO command, which is now obsolete.

The EIM system is used to define an OS/400 user from another authentication.

EIM can seek the OS/400 user from different sources:

- from the Kerberos authentication. This allows single signon (SSO)
- from SSL client certificate authentication

The [CFGEACAUTH](#) command defines which kind of authentication are valid.

```
Easycom EIM Configuration (CFGEACEIM)
Type choices, press Enter.
Easycom server library name . . . > EASYCOM Valeur alpha
Use EIM in Easycom . . . . . *YES *YES, *NO, *SAME
EIM valid from . . . . . *NONE HHMM =
EIM valid to . . . . . *NONE HHMM =
LDAP user for EIM . . . . . 'administrator'
LDAP password for EIM . . . . .
EIM logon is mandatory . . . . *NO *YES, *NO
LDAP dn for EIM . . . . .
LDAP service spn . . . . .
```

### Use EIM in EASYCOM

This is the main option for enable EIM on Easycom or not. Must be \*YES to enable the other options.

### SSO authorized from / SSO authorized to



EIM 'opening hours'. EIM connections are forbidden outside of those hours.

#### **LDAP user for EIM**

Local LDAP user. This username is required during a connection attempt, to retrieve the "OS/400" username associated to the "Windows" user name.

This local username is the name used when configuring EIM with iSeries Navigator (when selecting NetWork/EIM Domain Mapping/Domain Management/<yourDomain>).

You need to only put the username, not "cn=

#### **LDAP password for EIM**

This is the password for the local LDAP connection.

#### **EIM logon is mandatory**

Configures EASYCOM to deny all non-EIM connections (with username/password).

#### **LDAP dn for EIM**

This is an alternate way for giving LDAP logon name, allowing specific syntax. So this is valid only if user is left blank. A typical value is:

cn=

#### **LDAP service spn**

This allows a specific service principal name. If \*DFT is specified, Easycom calculates it using "krbsvr400" and the system name.

Example of valid values (with systemi5 name for the system, testdomain.com for the domain and TESTDOMAIN.COM for the realm):

krbsvr400/systemi5

krbsvr400/systemi5@TESTDOMAIN.COM

krbsvr400/systemi5.testdomain.com@TESTDOMAIN.COM (default if \*DFT is specified)

### **EASYCOM Exit Programs**

#### **Exit Programs**

EASYCOM offers many programs called « Exit Programs ».

Those kinds of programs follow a given specification and must be implemented by the administrator of the AS/400, allowing a most advanced control and security of easycom connections and usage.

Some of them must be written in some configuration cases like the exit programs related to [Single Sign On](#) or to the Easycom lock ([Lock EASYCOM Host](#)).

Others are related to a specific configuration but are not mandatory, like with [Prestarts Jobs](#) use.

The others are not mandatory and are designed to have better security and control.

Sample sources are provided in **EACSYSSRC** source file in Easycom library.

### **Easycom startup**

#### **Starting Client Job - EACSTART**

If a program named EACSTART exists in the job libraries list (LIBL), it is called each time EASYCOM client job is submitted.

It is particularly useful to set properties or perform maintenance actions.

This program is called when the user is known. It can still modify attributes or parameters but cannot cancel the job, except by hardly killing it.

EACTCP003 is to be preferred to control user rights and eventually cancel the job.



### Prestart job initialization - EACPJINI

If Pre-starts Jobs are activated in EASYCOM server configuration and if EACPJINI program exists in the job library list, it is called each time Pre-start Job is started by the system.

EACPJINI offers the possibility to define job properties when the job is created. At that time the connected user is unknown.

EACTCP002 will be called at connection time.

### SQL initialization - EACSQLINI

This exit program is called when Easycom is using the SQL interface for the first time in the job (between SQL initialization and actual SQL usage, like SQL query prepare)

If using pre-start jobs, it is called **before the connection is made (SQL is initialized at this moment to reduce the connection delay)**, during the pre-start process; otherwise, it is called when the SQL for the first time (so it is never called if SQL is not used by the application).

This exit program can be used to check the environment at this point.

### Logon and access security

#### Connection control - EACTCPP01

This exit program is designed to control the connection before any authentication. This can deny connection before any password or ticket exchange is made.

This can also be used to control whenever the connection must or can be made using SSL.

```
PGM PARM(&LIB &TPNAME &RMTADDR &IPVERSION +
&SSLASK &SSLCNF &VALID)
  DCL VAR(&LIB) TYPE(*CHAR) LEN(10)
  DCL VAR(&TPNAME) TYPE(*CHAR) LEN(30)
  DCL VAR(&RMTADDR) TYPE(*CHAR) LEN(50)
  DCL VAR(&IPVERSION) TYPE(*CHAR) LEN(1)
  DCL VAR(&SSLASK) TYPE(*CHAR) LEN(1)
  DCL VAR(&SSLCNF) TYPE(*CHAR) LEN(1)
  DCL VAR(&VALID) type(*CHAR) len(10)
```

**&LIB** is the library that when the Easycom program is. Usually, Easycom.

**&TPNAME** is the name of the Easycom program. By default, this is Easycom.

**&RMTADDR** is the TCP/IP address of the connection request. This can be in IPV4 or IPV6 form depending on &IPVERSION value.

**&IPVERSION** is equal to 4 or 6 depending on the IP version currently in use for the connection (if the AS/400 supports it, Easycom will accept both protocols by default)

**&SSLASK** informs if the client will try to negotiate an SSL connection. Possible values are:

- 'Y': the client supports SSL, and if the server accepts it, the connection will be made using SSL. In other words, the connection will maybe use SSL.

- 'N': the client is not supporting SSL or doesn't ask to use it. In other words, the connection won't use SSL in any case.

**&SSLCNF** informs if the SERVER will or supports SSL. Possible values are:

- 0: the server won't use SSL at all (even if supported)
- 1: the server may use SSL if SSLASK=Y. If SSL negotiation fails, the connection will remain valid.
- 3: the server will use SSL. If SSLASK=N or if the SSL negotiation fails, the connection will be aborted.

**&VALID** is used to tell EASYCOMD to grant or deny the connection. Possible values are:

- \*YES: the connection process can continue

- \*DENY: the connection is aborted immediately. An error message will be prompted on the client.

Note: only **&SSLCNF** and **&VALID** can be modified by the exit program.

### Logon control - EACLOG002

EACLOG002 is an exit program for general authentication process.

This program is called **after** the authentication made by Easycom.

This exit program is called on all authentication situations (normal, SSO, and EIM).

It can be used to audit the Easycom usage and/or deny connections from custom criteria.

EACLOG001 is the previous version of EACLOG002; it won't be called if EACLOG002 is implemented.

EACLOG002 has only two more parameters for IP version and SSL condition.

The prototype is:

```
PGM PARM(&LOGTYPE &RC &LOGUSER &LOGDOMAIN &USER
&IPADDR &STATION &IPVERSION &SSL)
DCL VAR(&LOGTYPE) TYPE(*CHAR) LEN(10)
DCL VAR(&RC) TYPE(*CHAR) LEN(10)
DCL VAR(&LOGUSER) TYPE(*CHAR) LEN(130)
DCL VAR(&LOGDOMAIN) TYPE(*CHAR) LEN(130)
DCL VAR(&USER) TYPE(*CHAR) LEN(10)
DCL VAR(&IPADDR) TYPE(*CHAR) LEN(130)
DCL VAR(&STATION) TYPE(*CHAR) LEN(130)
DCL VAR(&IPVERSION) TYPE(*CHAR) LEN(1)
DCL VAR(&SSL) TYPE(*CHAR) LEN(1)
```

**&LOGTYPE** is input, and tells which logon is being processed. The possible values are:

\*STD: this is a standard login/password logon (&LOGUSER and &LOGDOMAIN are not available)

\*EIM: this is an EIM logon. No password is available. &LOGUSER, &LOGDOMAIN and &USER are applicable.

\*SSO: this is an Easycom kind SSO. All fields are available.

**&RC** is the result of the command. This can be used to deny the user or indicate that the OS/400 user was changed.

The possible values are:

\*OK: the logon remains granted

\*CHG: the &USER parameter is changed by the exit program. Note: the &USER user will not have a password validation.

\*OUTOURS: the logon is rejected because of hours of work.

\*DENY: the logon is denied.

**&LOGUSER** is the Windows username. This is filled only in \*EIM or \*SSO mode for &LOGTYPE.

**&LOGDOMAIN** is the Windows domain. This is filled only in \*EIM or \*SSO mode for &LOGTYPE.

**&USER** is the OS/400 user. This is the OS/400 user under which the Easycom job will run.

**&IPADDR** is the IP address of the client connection. This can be used to filter access or for auditing.

**&STATION** is a string that represents the station of the client connection. This can be the real machine name (the name that corresponds to the IP address) or the Terminal name, if the connection is made thru an RDP connection.

**&IPVERSION** is equal to 4 or 6 depending on the TCP/IP network version used for connection. (IPv4 or IPv6)

**&SSL** is equal to 'Y' if the connection is using SSL and 'N' if not. SSL negotiation is already made currently.

### Security by restriction - EACTCP003

This exit program is designed for limiting EASYCOM use of to a user and/or a PC group.

If EACTCP003 program exists in EASYCOMD library list, it will be called at each connection attempt, excepted if EASYCOM is configured to use pre-starts Jobs (in this case [EACTCP002](#) can be used).

This program can allow or deny the connection from the client application.

If connection is accepted, it can submit by itself the client job or let Easycomd doing it.

**&JOBNAME** variable is used to determine what is decided:

- \*YES to accept the connection, but submit the job in the exit program.
- \*NO to refuse the connection
- Any value to let easycomd submit the job with that name.

Note: the initial value is equal to the jobname that is calculated during the connection, usually the name of the client pc if it is possible to use it as a jobname (or the jobname decided by the client application).

Program specification:

```
PGM PARM(&TPPGM &TPLIB &USER &EAC_PARM1 + &EAC_PARM2
&RMT_ADR &JOBNAME)
DCL VAR(&TPPGM) TYPE(*CHAR) LEN(10)
DCL VAR(&TPLIB) TYPE(*CHAR) LEN(10)
DCL VAR(&USER) TYPE(*CHAR) LEN(10)
DCL VAR(&EAC_PARM1) TYPE(*CHAR) LEN(30)
DCL VAR(&EAC_PARM2) TYPE(*CHAR) LEN(30)
DCL VAR(&RMT_ADR) TYPE(*CHAR) LEN(50)
DCL VAR(&JOBNAME) TYPE(*CHAR) LEN(10)
```

Parameters:

**TPPGM:** Target Program Name

**TPLIB:** Library containing TPPGM program.

Parameters TPPGM and TPLIB will be used by the EACTCP003 program if it submits the client job by itself.

**USER:** User name

New client connection username (can be used to limit access to a user group).

**EAC\_PARM1:** Parameter 1 of TPPGM program

First parameter to pass to target program (TPPGM) if EACTCP003 submits the client job by itself.

**EAC\_PARM2:** Parameter 2 of TPPGM program

Second parameter to pass to target program (TPPGM) if EACTCP003 submits the client job by itself.

**RMT\_ADR:** TCP/IP client address

TCP/IP client address (may concern workstations set).

**JOBNAME:** SBMJOB job name (Input / Output).

Name of the job to be activated in EASYCOM subsystem. Default name is the client station name.

On return, set JOBNAME parameter to:

\*NO, to refuse the connection.

\*YES, if EACTCP003 has submitted the client job by itself.

A name, or leave it unchanged, to accept the connection and let Easycom submit the client job.

#### Comments:

This exit program can be used to check the validity of the user id or TCP/IP address.

It can also submit the job under the authority of a user different from the one requesting the connection.

Or it can also submit another program, different from TPPGM, in order to setup some environment properties before calling TPPGM.

#### Prestart job control - EACTCP002

EACTCP002 works the same way as [EACTCP003](#) when Pre-starts Jobs are activated.

Since the job is already initialized, EACTCP002 does not create it but allows or refuses its start. It also permits controls or treatments prior initialization.

Note: this exit program is also called when not using prestart jobs.

#### 'Program Level' Security - EACP003

In addition to basic safety, **programs level** safety can be used.

Only programs validated by data processing department can be used on AS/400.

Unauthorized EASYCOM programs may be connected to AS/400 but will be unable to make any operation (file opening, program calling or other).

Authorized program will send a special password to EASYCOM. A data processing department AS/400 program returns information telling if password is accepted. This password can be similar, for example, to EASYCOM program coding.

#### To activate this mechanism:

If '**Lock EASYCOM host**' entry is set to \*YES in CFGEAC, no file can be opened, no program can be called, no command can be sent to AS/400 by EASYCOM, until the client application frees it sending a password to it.

This option requires writing an EACP003 script. This script must be in EASYCOM job LIBL.

Warning, if option is activated and script does not exist, EASYCOM will remain locked, and no job can be created.

Here is this script layout:

```
PGM PARM(&PASSW &RESULT)
DCL VAR(&PASSW) TYPE(*CHAR) LEN(100)
DCL VAR(&RESULT) TYPE(*CHAR) LEN(10)
...
/* IF PASSW HAS THE RIGHT VALUE */
CHGVAR VAR(&RESULT) VALUES('*YES')
...
/* IF PASSW DOES NOT HAVE THE RIGHT VALUE */
CHGVAR VAR(&RESULT) VALUES('*NO')
```

It receives a single-entry parameter (&PASSW applicative password different from profile password). It returns &RESULT parameter.

- \*YES value authorizes job starting and process to continue.
- \*NO value locks the job.

#### Easycom mode single signon - EACSSO001

It's the Exit Program associated to Single Sign-On activation in easycom mode. This is not called in EIM mode.

Note: this is recommended to use the EIM mode single signon instead of the Easycom mode.

When Single Sign-on is configured and activated (see [CFGGEACSSO](#)) and if program EACSSO001 exists in the job libraries list, it runs with various events:

- before memorizing a signature (simple connection or Windows session)
- when recording (simple connection or Windows session)

then with each connection request.

## Parameters

EASYCOM calls the program, transfers various parameters to it and turns over.

**&OP - Operation:** program call origin

*\*BEFORE and \*WINBEFORE*

Before memorizing simple or session signature, the program can:

- modify username and/or password
- allow or refuse memorizing

*\*SIGNON and \*WINSIGNON*

Signature memorizing, the program :

- can't modify user or password anymore,
- can allow or refuse memorizing

*\*REQUEST*

Requires connection, the program :

- cannot modify user or password anymore,
- can erase storage and force user to be signed again.

#### **&RC - Return**

\*OK: accepts signature

\*DENY: refuses signature

\*EXPIRED: signature validity period is exceeded

\*OUTHOURS: request out of authorized hours,

\*CHG: user change

#### **&USER / &USERLEN - user name length**

#### **&PWD / &PWDLLEN - password length**

&SOTIME – Time in HHMMSS format

&SODATE - Date in CYYMMDD format

&IDADR - IP client address

&STATION - workstation (different from &computer if TSE is used)

&COMPUTER – computer name

&LOGDOMAIN – Windows domain

&LOGUSER – Windows user

The fat variables (except &OP) can be modified with &RC program (to authorize or refuse signature or connection, change user, expiration or out of authorized domains), &USER and &PWD for a user change.

See EASYCOM library EACSSO001 file for an example and more detailed specifications.

### ***Objects and programs security***

#### **EACSOPEN - File open, SQL queries**

EACSOPEN exit program is called, if it exists in the client job LIBL, each time a file open is requested by the client job, or a SQL statement is prepared or immediately executed.

The exit program can refuse the file operation, or it can change file name or SQL statement.

See source example in EACSYSSRC file, Easycom library.

#### **EACSRCMD - Remote command**

EACSRCMD exit program is called, if it exists in client job LIBL, each time a command is submitted by the client application, with Easycom function API.

Exit program can refuse execution of the command, or it can replace the command before returning.

See source example in EACSYSSRC file, Easycom library.

#### **EACSCALL - Program Call**

Exit Program EACSCALL will be called, if it exists in the client job library list, each time an external program or procedure is called by the client application, using Easycom API.

This exit program can refuse the program or procedure call by the client application.

It can also change the program name, library name or procedure name on return, so that the client application will call another program.

See source example in EACSYSSRC file, Easycom library.

### EACSIFS - IFS access

This exit program is called on each IFS file open.

The parameters are the file path and open mode. The open mode is a numeric value that is a combination of the following constants (hexadecimal):

- \_EAC\_IFSOPEN\_READ=1 read access
- \_EAC\_IFSOPEN\_WRITE= write access
- \_EAC\_IFSOPEN\_CREAT=4 file will be created if not exist
- \_EAC\_IFSOPEN\_EXCL=8 file must not exist before open (create is mandatory)
- \_EAC\_IFSOPEN\_TRUNC=10 truncate file
- \_EAC\_IFSOPEN\_APPEND=20 append file
- \_EAC\_IFSOPEN\_BINARY=40 binary mode
- \_EAC\_IFSOPEN\_BIGFILE=big file. Allows to open > 2Gb files.

Create mode:

- \_EAC\_IFSMODE\_RUSR 400 user can read (u+r)
- \_EAC\_IFSMODE\_WUSR 800 user can write (u+w)
- \_EAC\_IFSMODE\_XUSR 1000 user can execute (u+x)
- \_EAC\_IFSMODE\_RGRP 2000 group can read (g+r)
- \_EAC\_IFSMODE\_WGRP 4000 group can write (g+w)
- \_EAC\_IFSMODE\_XGRP 8000 group can execute (g+x)
- \_EAC\_IFSMODE\_OTH 10000 others can read (o+r)
- \_EAC\_IFSMODE\_WOTH 20000 others can write (o+w)
- \_EAC\_IFSMODE\_XOTH 40000 others can execute (o+x)

Share mode:

- \_EAC\_IFSSHARE\_RDONLY 100 0000 read only share
- \_EAC\_IFSSHARE\_WROONLY 200 0000 write only share
- \_EAC\_IFSSHARE\_NONE 400 0000 no share (exclusive)
- \_EAC\_IFSSHARE\_RDWR 300 0000 read/write share.

If you need to test the open mode, you need to use a bitwise AND with the flag to test and see if the result is equal to that flag.

Note: the exit program can only deny or accept the file open.

A source sample is available in the EACSYSSRC file in EASYCOM library.

## Native programs and Data Queues

### AS/400 native program description

EASYCOM enables AS/400 native programs calling, CL or RPG programs or stored procedures.

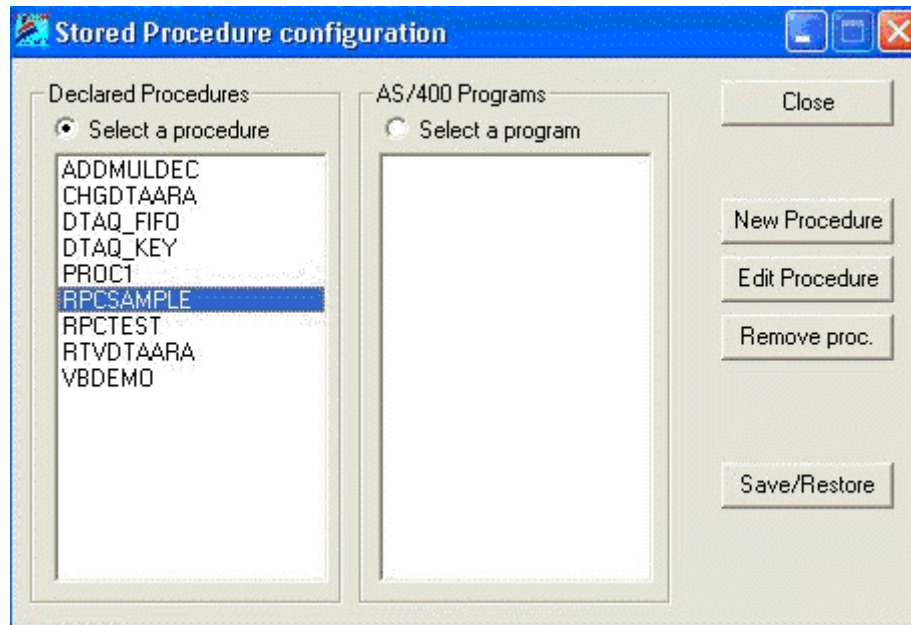
To perform this, EASYCOM needs these programs description stored on AS/400 in YPROCHDR and YPROCparms files in EASYCOM library.





Programs description and data queues are built by DTAQ-RPC constructor. The basic principle is to specify all parameters, types and uses (input, output, input/output) required to call the program.

The first screen displays the existing procedures (stored on AS/400) and enables to create, modify, or delete them. The descriptions can be saved in a PC text file in view of a later transfer to another AS/400.



A new name is assigned to the procedure. It does not need to match with the associated program name.

A native AS/400 program (CL, RPG, COBOL, C etc.) is associated to the procedure.

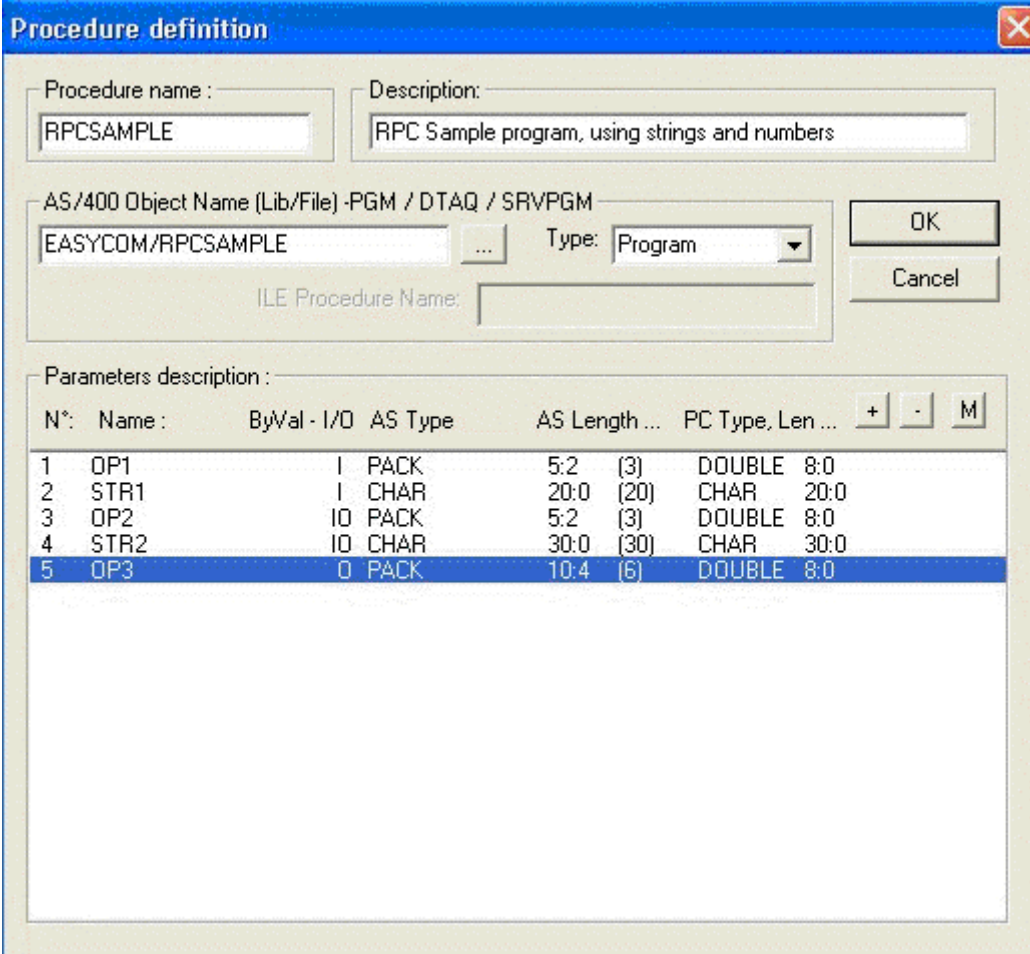
The library may be omitted or replaced by \*LIBL.

The description is a free text, which will be seen when client workstations browse through the procedures.

Each program calling type and size parameters are described.

Each parameter may be considered as a database table field.





**Procedure definition**

Procedure name :  Description:

AS/400 Object Name (Lib/File) -PGM / DTAQ / SRVPGM  
 ... Type:

OK Cancel

ILE Procedure Name:

Parameters description :

N°	Name :	ByVal	I/O	AS Type	AS Length ...	PC Type, Len ...	+	-	M
1	OP1		I	PACK	5:2 (3)	DOUBLE 8:0			
2	STR1		I	CHAR	20:0 (20)	CHAR 20:0			
3	OP2		IO	PACK	5:2 (3)	DOUBLE 8:0			
4	STR2		IO	CHAR	30:0 (30)	CHAR 30:0			
5	OP3		O	PACK	10:4 (6)	DOUBLE 8:0			

Each parameter can be considered as a field in a database table.

It therefore has a name, by which it can be referred to by the application.

Parameters designed to provide values for the called program are considered as input parameters (IN).

Parameters designed to receive a value on returning from the call are considered as output parameters (OUT).

Parameters that are modified by the program are both input and output (IN/OUT) parameters.

By default, all the parameters in an AS/400 program are both input and output. The logic of the program can change this property.

If a calling parameter of the program is a structure (DS : Data Structure), each field of the DS has to be described individually.

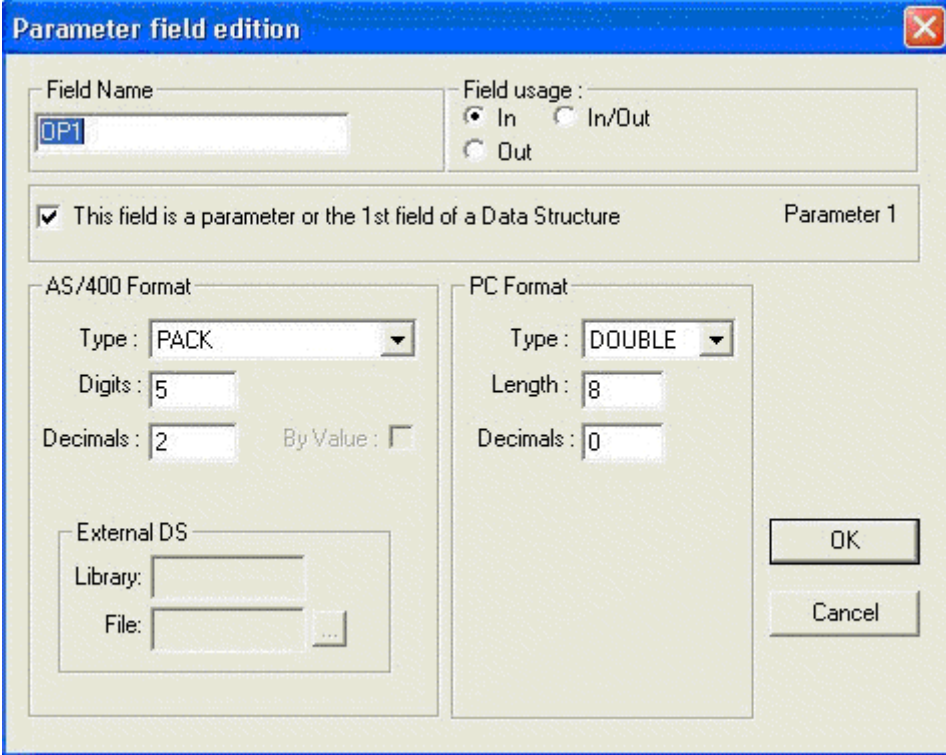
For the first field only, the box to be ticked is: This field is a parameter or the 1st Field.

The type of parameter expected by the AS/400 program must be specified exactly:

CHAR : Character data type.

BIN2 : 16-bit numeric data type.

BIN4 :	32-bit numeric data type.
PACK :	Condensed numeric data type (DECIMAL). This is the format in which CL handles numerical data (CL *DEC type).
ZONED :	Extended numeric data type (NUMERIC).
DATE :	AS/400 date in the yyyy-mm-dd format.
TIME :	Time in hh:mm:ss format.
FLOAT :	Numeric value in single-precision floating point.
DOUBLE :	Numeric value in double-precision floating point.
TIMESTP :	Elapsed time field.
GRAPHIC :	Character type data, not to be converted.
EXTERNAL DS :	A structure described by an external data structure, i.e., a physical file.



### *Migrating procedures and DTAQ from AS/400 to another*

When a developer works on an AS/400, he creates procedures and data queues which will subsequently have to be used on another AS/400, these procedures descriptions and data queues have to be transferred to the other AS/400.

Descriptions are stored in three files: YPROCHDR, YPROCPARMS and YPROCPGM. They are stored in AS/400 EASYCOM library (default). They can also be placed in another library. In this case they will be searched in connected profile LIBLE.

If two AS/400s are connected, the above three files can obviously be transferred directly from one to the other.

Otherwise, the DTAQ-RPC manufacturer offers a facility to import/export descriptions from or to text files, that means that the necessary descriptions can be saved on the developer's workstation and then restored on the client's.

### Calling a ILE procedure

The ILE procedure must be described, as a program (Type \*PGM).

The AS/400 object in "AS/400 Object Name" field must be \*SRVPGM type.

It must be "Service Program" type.

The first described parameter is the procedure returned value.

Only returned values type "Integer 32 bits" are accepted.

Then, the parameters are described, as for a OPM program.

16 parameters maximum are accepted for the procedure, plus returned value.

**Procedure definition**

Procedure name : PROC1

Description:

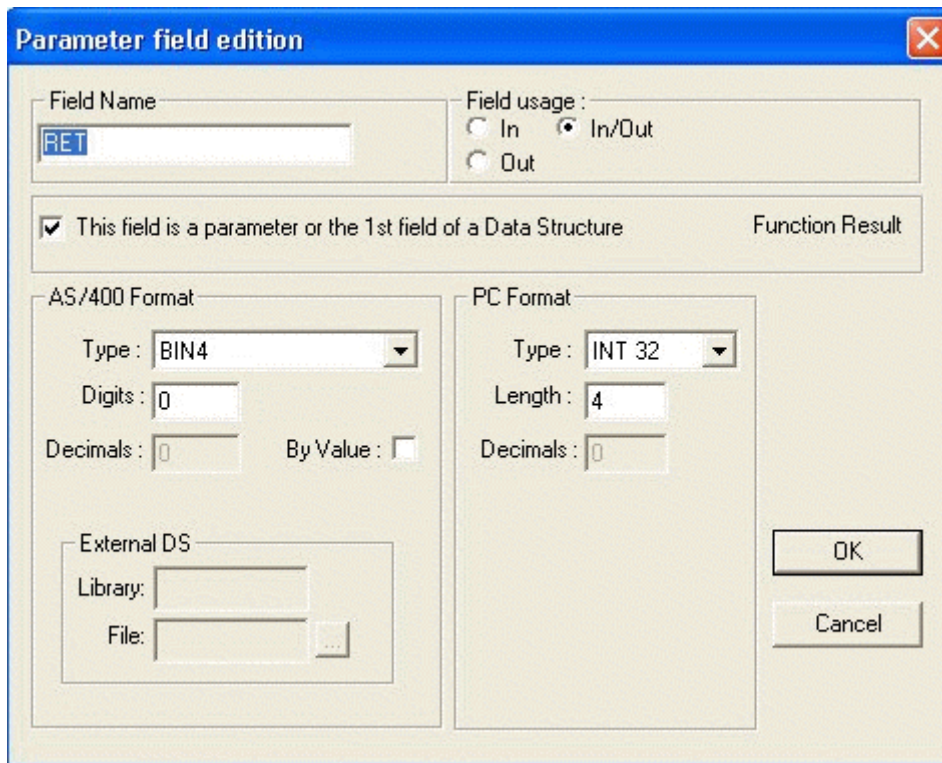
AS/400 Object Name (Lib/File) : AURA/SRVPGM01 ... Type: Service Program

ILE Procedure Name: Proc1

OK Cancel

Parameters description :

N°	Name	ByVal	I/O	AS Type	AS Length	PC Type	Len
Ret	RET			IO BIN4	0:0 (4)	INT 32	4:0
1	P1			IO BIN4	0:0 (4)	INT 32	4:0
2	P2			IO BIN4	0:0 (4)	INT 32	4:0
3	P3			IO BIN4	0:0 (4)	INT 32	4:0
4	P4			IO BIN4	0:0 (4)	INT 32	4:0
5	P5			IO BIN4	0:0 (4)	INT 32	4:0
6	P6			IO BIN4	0:0 (4)	INT 32	4:0
7	P7			IO BIN4	0:0 (4)	INT 32	4:0
8	P8			IO BIN4	0:0 (4)	INT 32	4:0
9	P9			IO BIN4	0:0 (4)	INT 32	4:0
10	P10			IO BIN4	0:0 (4)	INT 32	4:0
11	P11			IO BIN4	0:0 (4)	INT 32	4:0
12	P12			IO BIN4	0:0 (4)	INT 32	4:0
13	P13			IO BIN4	0:0 (4)	INT 32	4:0
14	P14			IO BIN4	0:0 (4)	INT 32	4:0
15	P15			IO CHAR	200:0 (200)	CHAR	200:0



For each parameter, option "By Value" has been added.

This option is valid only for parameters type "32 Bits integer".

When checked, this option indicates that the procedure receives this parameter in value, and not in address.

## TroubleShooting

### How to diagnose errors

In case of [connectivity errors](#), there are several ways to search:

- in the EASYCOMD job history. To see it, do WRKACTJOB, then option 5 on EASYCOMD, then option 10 (job's history), and type F10. Type F1 on the suspicious messages to get more information.
- In the EACMSGQ messages. To see it, enter DSPMSG EASYCOM/EACMSGQ.
- In the QSYSOPR messages. To see it enter DSPMSG QSYSOPR. Unexpected, failures or licensing messages will appear here.
- In the QEZJOBLOG OutQ. A spool file is generated in this outq if Easycom failed to start properly (error -4 on the client), or if the easycom job unexpectedly stops.

To see the spool file, do the following commands:

- WRKOUTQ OUTQ(QEZJOBLOG)
- Type F18 (to go at the end), and then F11.
- There should be a line with the station name, with the corresponding user, date and time.

Type 5 on then entry to display the spool. It contains information, warning and error messages of the job.

- In the LOGFILE file, LOGFILE member in the EASYCOM library. This file will contain all TCP/IP failures, with OS/400 errors codes. To see that file, use: DSPPFM EASYCOM/LOGFILE MBR(LOGFILE)

This file can be downloaded using FTP or [Easycom configuration](#) client on Windows.

**In case of errors during processing**, an Easycom logfile can be useful. It can be setup using [CFGEAC](#) or [Easycom configuration](#). The contents of this file can help to understand what is performed on the server, see parameters, additional error messages, ...

The Easycom job history can also help a lot. To see it, use WRKACTJOB, option 5, then 10 and type F10.

If the Easycom job stops too quickly to see a job history, use JOBLOG(\*YES) in [CFGEAC](#) command to setup EASYCOM to always have a spool generated in QEZJOBLOG (see above to consult it).

If the Easycom job aborts unexpectedly there should be a spool file in QEZJOBLOG (see above).

If the EASYCOM or the EASYCOMD job fails in a loop, try to see what is involved in the call stack. It is available by using WRKACTJOB, then option 5, then option 11.

**In case of licensing errors**, do DSPMSG QSYSOPR if the information provided on the client is not sufficient.

### *How to contact our Technical Support ?*

Contact EASYCOM Technical support.

E-Mail : [tech@easycom-aura.com](mailto:tech@easycom-aura.com)

Internet : <http://www.easycom-aura.com/>

### *Important before contacting us ...*

For quick and efficient answers from our technical support to your queries, you are kindly requested to prepare your call as follow:

- Complete and return EASYCOM registration card (AURA EQUIPEMENTS yellow card).
- Enter the product activation key on AS/400.
- Perform several tests to define the original problem.
- Identify EASYCOM version.
- **Check that your Premium Assistance contract is valid.**
- Make a note of the local or network hardware and software configuration, and the PC's configuration where the problem had occurred.
- Make a note of all recent configuration modifications.

- Make a note of the different tested operations and displayed error messages.
- Consult FAQ (in the help online).

**If you have not found solution, can contact us :**

Mail: tech@easycom-aura.com

### *Maintenance agreement*

AURA Equipements offers several technical support levels.

On request, we will provide you with the best commercial offer matching your needs.

For general or commercial information, contact: info@easycom-aura.com.

### *AS/400 trace file*

#### *EASYCOM trace file activating*

Use EASYCOM Configuration utility, trace files management bookmark. Library and traces file name to be created, its detail level (from 1 to 9, level 1 is basic, level 9 is the most detailed) can be set.

Option "Time print" allows having a timestamp in front of each operation line.

This trace can be retrieved on PC from the same screen as a text file.

This trace can also be activated from a terminal with CFGEAC command.

#### *Easycom log file*

Trace file enables EASYCOM carried out operations to be displayed on client or server side. AS/400 EASYCOM server processes elementary requests applied to tables or procedures.

It receives a process request from the network and returns a response.

The requests and responses are recorded in a trace file, it can be used as basis to analyze data flow between client and server.

Lines starting with << indicates client request.

Lines starting with >> indicates AS/400's answer.

```
<<EACopen(EASYCOM/SP_CUST,4194309,-1)  B Requête.
9 Fields, 0 key fields
EAC_NO_CVT - mode=
>>Ret=1; Err=0; Msg=; Int=0  B Réponse.
```

In AS/400 trace, if time option was selected, all requests and responses are time in hh:mm:ss.ms format.

```
<<15:48:45.566: EACread(1,p(2275),91,34144281,(null),0,p(100))
```

In response, data "Clk=x" indicates AS/400 CPU time spend to process the request.



>>15:48:45.574: Clk=8, Len=619; Ret=5; Err=0; Msg=; Int=

## Trace file header (common to all sessions)

This trace file part is always the same for all EASYCOM sessions.

**Time is : 03/27/2000 - 17:22:10**

**Easycom Server Version is : 4.5712, Link is TCP/IP**

**Client license is : D\$WINDEV10 , Easycom Library is : EASYCOM**

**JobName=ALBATROS, User=QPGMR , QCCSID=297 Heart Beat freq :10**

**Easycom Log File TRACE/SR, level 1**

-----

>>Ret=1; Err=0; Msg=; Int=0

<<RTV\_AS\_VER(p(4))

>>Ret=4; Err=0; Msg=; Int=0

<<WriteTableEBCDI(49 42 4D 43 43 53 49 44 20 30 20 31 32 35 32 00 00 00 00 00 ... (256))

**Build Table from CCSID:0 to 1252**

**open IBMCCSID01252, IBMCCSID000000000100**

<<WriteTableASCII(49 42 4D 43 43 53 49 44 20 31 32 35 32 20 30 00 00 00 00 00 ... (256))

**Build Table from CCSID:1252 to 0**

**open IBMCCSID00000, IBMCCSID012520000100**

<<ReadTableASCII(p(256))

>>Ret=0; Err=0; Msg=; Int=

<<ReadTableEBCDIC(p(256))

>>Ret=0; Err=0; Msg=; Int=0

<<EACSqlDeclare(2A 45 41 43 20 43 56 54 20 4E 4F 00 ,12)

**Statement:\*EAC CVT NO**

>>Ret=1; Err=0; Msg=; Int=0

<<EACSqlBegin()

>>Ret=0; Err=0; Msg=; Int=0

## Physical file opening trace

```
<<EACopen(EASYCOM/SP_CUST,4194309,-1)
9 Fields, 0 key fields
EAC_NO_CVT - mode=rr+
>>Ret=1; Err=0; Msg=; Int=
<<EACgetdesc(1,p(65000),65000,939786240,(null))
>>Ret=9; Err=0; Msg=; Int=
```

## Logical file opening trace

```
<<EACopen(EASYCOM/SP_CUST_UN,4194309,-1)
LF with 1 Data Members, 1 Record Formats
9 Fields, 1 key fields
EAC_NO_CVT - mode=rr+
>>Ret=2; Err=0; Msg=; Int=
<<EACgetdesc(2,p(65000),65000,939786240,(null))
>>Ret=9; Err=0; Msg=; Int=
```

## File records reading trace

```
<<EACread(2,p(816),102,34144264,(null),0,p(32))
```

**VERB=\_EAC\_NEXT LOCK=OFF RECS=8 FILE=EASYCOM/SP\_CUST\_UN**

**RRN=2 RRN=4 RRN=5 RRN=6 RRN=7 RRN=8 RRN=9 RRN=10**

```
>>Ret=8; Err=0; Msg=; Int=0
```

Read operation type is indicated by « VERB=

Where xxxx may be :

FIRST, NEXT, PREV, LAST, KEY\_EQ, KEY\_GE, KEY\_GT, ...

"**LOCK=**" indicates if operation is carried out with or without record locks.

"**RECS=**" indicates maximum records number requested for the response.

This number is directly linked to "Records= data" in the "Easycom.ini" file "Buffers section" on client PC.

"**RRN=**" indicates the records read number.

In response, "Ret=n" indicates the records number actually returned, to the read request.

If the read operation fails because of an input/output error, the message is stored in the trace, and the records read are returned.



```
<<EACread(2,p(3570),102,34144291,(null),0,p(140))
```

```
VERB=_EAC_NEXT LOCK=OFF RECS=35 FILE=EASYCOM/SP_CUST_UN
```

```
RRN=11 RRN=12 RRN=13 RRN=14 RRN=15 RRN=16 RRN=17 RRN=18 RRN=19 RRN=20  
RRN=54 +...
```

```
... RRN=55 RRN=56
```

```
**SIGIO** Msg:CPF5001
```

```
>>Ret=13; Err=5001; Msg=CPF5001; Int=0
```

In this example, 35 records are requested, but only 13 are available until file end.

To obtain the detailed error message, use [DSPMSGD](#) command.

## SQL request opening trace

```
<<EACopen(SELECT * from SP_CUST where LASTNAME>'M',4194309,-1)  
Statement : SELECT * from SP_CUST where LASTNAME>'M'  
Cursor 0  
>>Ret=2; Err=0; Msg=; Int=  
<<EACgetdesc(2,p(65000),65000,939786240,(null))  
>>Ret=9; Err=0; Msg=; Int=
```

### Error codes

#### TCP/IP Errors

Negative error codes mean an Easycom protocol error during TCP/IP connection, and positive ones mean native TCP/IP errors.

Native codes (positive) change depending on the client platform (Windows, Linux, AIX, iSeries, ...).

All errors come with a local error text, and most of the time with a specific error text coming from the iSeries.

Here are negative codes :

Error code	Description
-1	Error while submitting the job. SBMJOB made by the EASYCOMD job failed. Additional error text coming from iSeries will is provided with this error. The EASYCOMD job history should contain all information on that failure.
-2	Security not valid. This error can occur if wrong user, password, password disabled, etc. The detailed reason is specified as text.
-4	submitted job did not answer, or failed to initialize data queues The most common reason for this is that the job failed to run. It was submitted, but ended before beginning to communicate with the client. This is usually caused by wrong user's jobd. The full reason can be found in the QZJOBLOG OUTQ of the system. To see it, do the

	<p>following commands:</p> <ul style="list-style-type: none"> <li>○ WRKOUTQ OUTQ(QEZJOBLOG)</li> <li>○ Type F18 (to go at the end), and then F11.</li> <li>○ There should be a line with the station name, with the corresponding user, date and time.</li> <li>○ Type 5 on it to see the errors.</li> </ul>
-5	Password is expired. If the client program 'catches' this error, it can perform a custom password change dialog box, and send the password with the new connection request. The password send by the application will have the following form in this case: oldpassword@newpassword
-6	Internal reject 1. Unexpected error, caused by a bug in EASYCOMD. Please contact help support. Restarting EASYCOM subsystem is recommended.
-7	Failed to init the library list. Errors occurred when installing the libraries that are defined in the user's jobd. You can consult the QEZJOBLOG outq for more information (see error -4)
-8	SSO error. SSO profile is expired (re-signoon required), or not supported by EASYCOMD.
-9	The server cannot accept Kerberos tickets. EIM SSO is not configured, or the EASYCOMD LDAP connection failed. Do DSPMSG EASYCOM/EACMSGQ to see if there are Kerberos-related messages. Check that you see 'Eim=
-10	Timeout on read. Communication error: the read request timed out. The connection was probably broken.
-11	Logon cancelled. This error occurs when message boxes are enabled and when the user clicks on 'cancel'.
-12	Connection broken. The connection was broken by peer.
-13	Kerberos negotiation protocol failure. There was an unexpected Kerberos error when connecting. Check EIM configuration, and check if the same user is working using IBM Client Access in EIM mode.
-14	Kerberos error on client. The client failed to generate a ticket to send to the server. Additional text should explain the reason.
-15	Kerberos error on server. The server did not recognize the ticket or failed to grant it.
-16	OS/400 incompatible version. The OS/400 version is not compatible with the current request.
-17	Unexpected error while submitting (state unknown). Unexpected error probably caused by a bug. Please contact help support.
-18	Kerberos authentication out of hours. See CFGEACSSO to setup the valid hours for Kerberos authentication.
-19	Out of hours by exit program. The EACLOG001 exit program returned that the login is not valid at this time.
-20	Denied by exit program. The EACLOG001 exit program return that the login is denied
-21	Not processed by exit program. The EACLOG001 exit program returned that the login is invalid.
-22	Kerberos authentication is not supported by the server. See that CFGEACSSO enabled *EIM mode and that EASYCOMD started properly (DSPMSG EASYCOM/EACMSGQ).
-23	Kerberos authentication is mandatory. The Easycom server was configured to accept only Kerberos authentication, but a regular login was attempted.
-24	Failed to use the target library. The library specified by the target program property (Program= in easycom.ini, in section [general]) was not usable, because nonexistent or other reason.
-25	Awake on private job failed. The application attempted awaking a job that was registered by the setting, but it fails. A new connection is required. Note: this error currently can appear only with

	Easycom For PHP.
-26	SSL required on this server. The SSL negotiation was not setup or failed, but is required on the server. This can be marked as required using the <a href="#">EACTCPP01</a> exit program or using CFGEAC command.
-27	SSL server error. The SSL negotiation failed because the server detected an error. There are probably some information in the EACMSGQ message queue (type DSPMSG EASYCOM/EACMSGQ on a terminal)
-28	SSL negotiation was made, but a failure is detected while passing the connection to the Easycom job.
-29	SSL client error. The SSL negotiation failed because of an error on the client. More additional information is provided in the error message text.
-30	SSL sequence error. The SSL negotiation sequence was detected as invalid
-31	SSL protocol error. An SSL error is detected during SSL handshake.
-32	SSL error: SSL not supported on the platform
-33	EIM was mandatory for login
-34	SSL authentication is mandatory
-35	SSL authentication error (bad certificate, expired, ...)
-36	EIM error
-37	No valid authentication provided. This means that all kind of accepted authentication methods failed.

Here are most common TCP/IP error codes:

Windows error code	AS/400 error code	Description
<a href="#">10061</a>	ECONNREFUSED 3425	Connection refused. No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host - i.e. one with no server application running.
<a href="#">10060</a>	ETIMEDOUT 3447	Connection timed out. A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.
<a href="#">11001</a>	HOST_NOT_FOUND 5 (host error category)	Host not found. No such host is known. The name is not an official hostname or alias, or it cannot be found in the database(s) being queried. This error may also be returned for protocol and service queries, and means the specified name could not be found in the relevant database.
10053	ECONNABORTED 3424	Connection aborted. An established connection was aborted by the software in your host machine, possibly due to a data transmission timeout or protocol error.
10064	EHOSTDOWN 3428	Host is down. A socket operation failed because the destination host was down. A socket operation encountered a dead host. Networking activity on the local host has not been initiated. These conditions are more likely to be indicated by the error WSAETIMEDOUT.

10050	ENETDOWN 3433	Network is down. A socket operation encountered a dead network. This could indicate a serious failure of the network system (i.e. the protocol stack that the WinSock DLL runs over), the network interface, or the local network itself.
-------	---------------	--

The localized error text is available at runtime and can be shown by the application or Easycom dialog boxes.

NB: most connection problem are caused by routers or firewalls installed on the client stations, or on the network.

Easycom is using one TCP/IP connection by default on tcp port 6077.

### Internal Errors

Error code	Description
257	You may not open another file then specified for the demonstration
258	License key is not valid. Please do DSPMSG QSYSOPR to have full details if needed (which kind of license is required)
260	License key expired. Please do DSPMSG QSYSOPR to have full details if needed (which kind of license is required)
261	No free connection. The number of allowed simultaneous connection was reached, and this new connection is not allowed.
263	License key not found. There is no license for the product currently used. Please do DSPMSG QSYSOPR to have full details if needed (which kind of license is required)
275	There is no license for this option. An option of the product is required but not found. DSPMSG QSYSOPR can contain more information if needed.
1	Parameter error. There was an invalid request sent to Easycom. This can be caused by unexpected usage of Easycom, a bug in the application or a bug in the Easycom upper stack (specific part to a product, like Delphi, WinDev, PHP, ...)
2	Memory allocation error. This is usually caused by incorrect size during memory allocation on the server. The possible reasons are: unexpected usage of Easycom, a bug in the application or a bug in the Easycom upper stack (specific part to a product, like Delphi, WinDev, PHP, ...)
3	File not opened. Attempt on a non-opened file. This is probably an application or easycom bug.
522	Cannot convert a NULL parameter. Problem during iSeries <-> client conversion on a NULL value.
527	Problem during ALCOBJ action. ALCOBJ was requested but failed
528	Failed to create an object. An object creation attempt failed.
529	Timeout on pgm call. A timeout was defined for a pgm call (using CFGEAC or by client application), and this timeout was reached. The program call was cancelled, with possible non closed context. Restarting the connection is recommended.
530	Procedure not found. A procedure call was requested, but the procedure was not found in the service program

**Error 10060 (3447 in Unix): Connection Timed out**

The called TCP/IP address does not exist on the network.

AS/400's TCP/IP address or name must be checked.

If an AS/400 machine name is used, check that it is properly referenced on DNS servers.

**Error 10061 (3425 in Unix): Connection Refused**

IP address or name of AS/400 must be checked.

EASYCOM system proper launch on AS/400 must be checked.

Subsystem must be launched with command:

**STRSBS EASYCOM/EASYCOM**

If the subsystem was started, check if EASYCOMD job runs.

If not, it must be started with the command:

**STREACD EASYCOM**

Or, subsystem must be stopped restarted.

Connection must be tested using EASYCOM configuration or administration tool.

If the EASYCOMD job can't be started, messages that EASYCOM generate have to be checked using the commands:

**DSPMSG EASYCOM/EACMSGQ** or **DSPPFM EASYCOM/LOGFILE**

Default EASYCOM port number is 6077.

If this number is already used, use [CFGEACTCP](#) to configure another port, and change [client configuration](#) to select the port number.

**Error 11001: (Host error 5 in Unix) Host not found**

On TCP/IP network AS/400 can be identified with its name at DNS level or host file. This error occurs when this name is used as IP address in connection parameters and is not found and associated with the right IP address.

AS/400 name, host file, DNS servers, must be checked or an IP address in xxx.xxx.xxx.xxx format must be used.

**Where is the Hosts file located?**

Usually in C:\WINDOWS\system32\drivers\etc repertory.

This file contains IP addresses relation to host names. Each entry must set on his proper line. The IP address must be placed in the first column, followed by the related host name. The IP address and the host name must be separated with one space at least.

Moreover, comments can be inserted on their proper lines or after computer name. They are indicated with '#' symbol.

Example:

194.206.10.1 main.as # main AS/400 server  
194.206.10.2 test.as # AS/400 test server  
194.206.10.100 serveur.info1  
194.206.10.101 poste\_x  
....

## DNS Server

Allows checking a DNS server address in connection network Internet (TCP/IP) Protocol properties.

## Product licensing

### *Registration card*

To get your product activation key and take advantage of the warranty, please complete and return EASYCOM registration card (AURA EQUIPEMENTS yellow card).

### *Moving licenses to new hardware*

If you change your iSeries, the activation key(s) that you have will no longer be valid for the new machine.

Every license is granted for a specific company and a specific iSeries. To make this change, print out, complete, and return to us the following form:

[http://www.easycom-aura.com/doccom/attestchange\\_vf.pdf](http://www.easycom-aura.com/doccom/attestchange_vf.pdf)

When you have installed and tested the new iSeries machine, you need to uninstall EASYCOM from the old iSeries.

### *What needs to be uninstalled on the old iSeries ?*

You need to delete the EASYCOM library by the following commands:

```
ENDSBS EASYCOM *IMMED  
DLTLIB EASYCOM
```

### *Copyright*

The information contained in this document can be modified without prior notice and does not engage AURA Equipements. The Software described in this document is governed by a licensing or agreement of confidentiality. The software cannot be used, copied or reproduced on any support according to the terms of this license or this agreement of confidentiality. No part of this handbook can be reproduced or transmitted by any manner, electronic or mechanical, including by photocopy or recording, without the express and written permission of AURA Equipements.

© 1986 - 2024 AURA Equipements. All rights reserved.

IBM, PC/AT, AS/400, iSeries, System i, i5/OS, power 5, PASE, AIX are trademarks of International Business Machines Corporation.

Windows, Windows Mobile, Word, Excel, Office 400 are trademarks of Microsoft Corporation.

EASYCOM and LAUNCHER 400 are trademarks of Aura Equipements.



All the quoted marks are trademarks by their authors.

## EASYCOM Configuration

### The PC tool

A PC tool is provided to centralize and manage the AS400 native access of the EASYCOM server. All options selected within the PC tool are stored into the **easycom.ini** file on Windows.

This configuration file can be general (in Windows directory, C:\WINDOWS) or be specific to an application (in the executable directory).

Unix versions (AIX, Linux or other) use the /etc/easycom.conf file, with the same syntax.

Using this file is optional for deployment. This is used only for convenience, allowing to avoid having connection parameters managed by the application program itself.

This tool contains the following tabs:

- Connection parameters
- EASYCOM Activation key
- Trace file
- Default settings
- Security
- Checking installation and versions

### Connection parameters

#### AS/400 name or IP address

Machine name or TCP/IP address for the AS/400 is entered here.

Use of a name implies a DNS configuration or hosts' file.

The **port number** can be specified with ":portnum", for example: iseries:6078 to have 6078 port number.

The **easycom** service name is used to setup the default port; and if no service is defined, the 6077-port number will be used.

#### EASYCOM Server

- **Default (EASYCOM/EASYCOM)**

Use the default server program: (EASYCOM/EASYCOM)

- **Other**

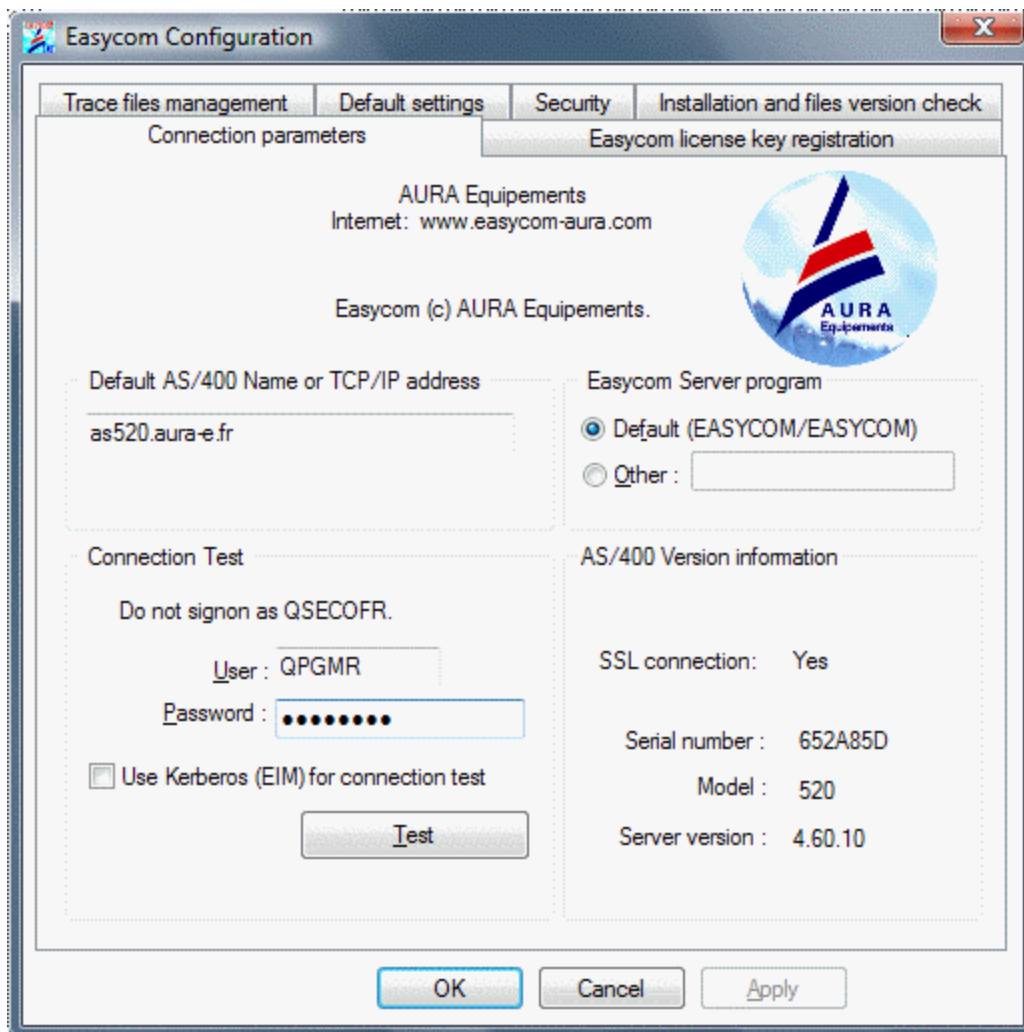
Select the server program to be activated (LIBRARY/PROGRAM) during connection. The server program is an AS/400 program started by the router or started by EASYCOMD job.

If no library is given, the library where EASYCOMD is running will be used.

#### Connection test

These options are used only for connection test and are not saved in the configuration file.





**Easycom Configuration**

Trace files management | **Default settings** | Security | Installation and files version check

Connection parameters | Easycom license key registration

AURA Equipements  
Internet: www.easycom-aura.com

Easycom (c) AURA Equipements.

Default AS/400 Name or TCP/IP address  
as520.aura-e.fr

Easycom Server program  
☒ Default (EASYCOM/EASYCOM)  
☐ Other :

Connection Test  
Do not signon as QSECOFR.  
User : QPGMR  
Password :   
☐ Use Kerberos (EIM) for connection test  
Test

AS/400 Version information  
SSL connection: Yes  
Serial number : 652A85D  
Model : 520  
Server version : 4.60.10

OK Cancel Apply

Click on **"Test"** button.

If connection is successful, AS/400 version information's are displayed, as for example:

Serial number: 650643C

Model: 520

Server version: 4.60.10

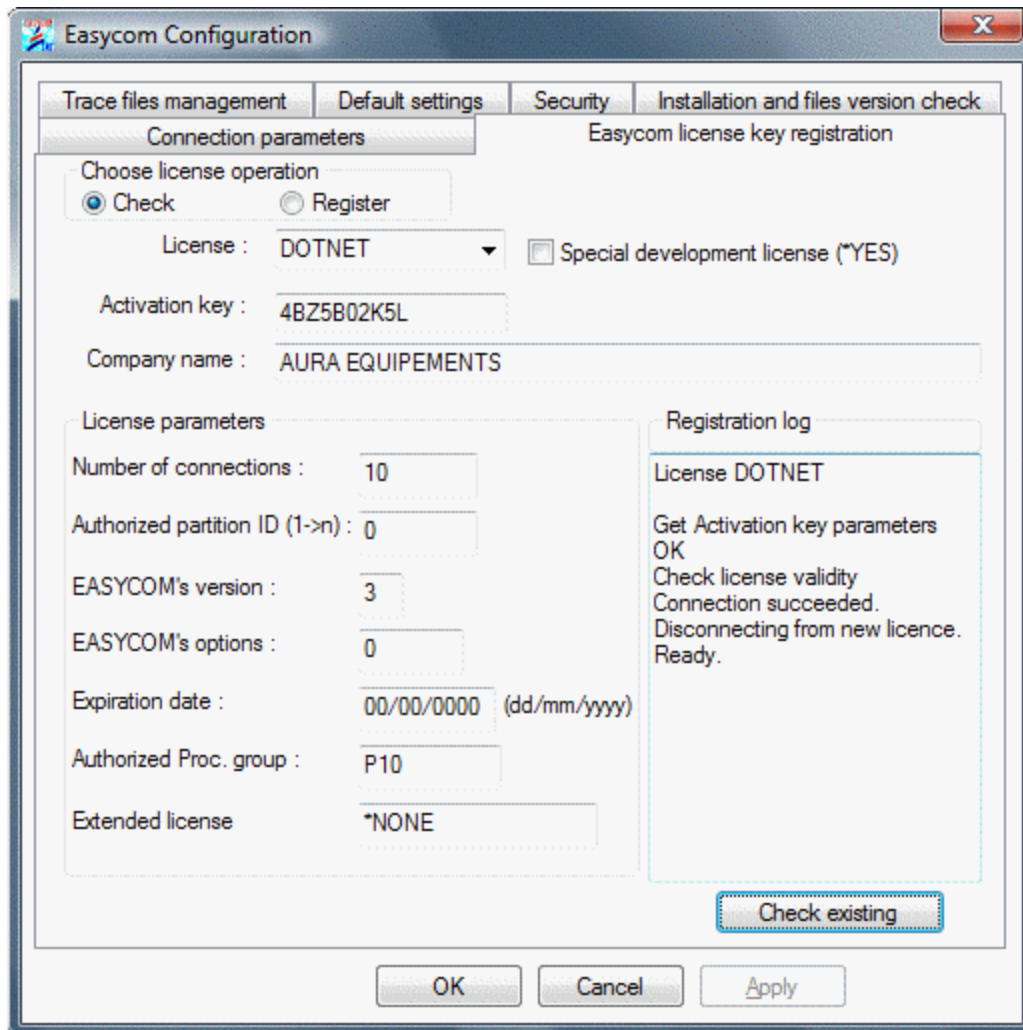
If SSL connection was setup this will show if the connection was actually in SSL.

If Kerberos connection was selected the actual OS/400 username will be shown in the information box.

### Easycom license key registration

Activation key is provided by AURA Equipements. If this is a purchased product, the registration card will be claimed to obtain the activation key. For evaluation process, the activation key is automatically sent after having downloaded the product.





**Easycom Configuration**

Trace files management | **Default settings** | Security | Installation and files version check

Connection parameters | Easycom license key registration

Choose license operation  
☒ Check ☐ Register

License : DOTNET ☐ Special development license (\*YES)

Activation key : 4BZ5B02K5L

Company name : AURA EQUIPEMENTS

License parameters

Number of connections : 10

Authorized partition ID (1->n) : 0

EASYCOM's version : 3

EASYCOM's options : 0

Expiration date : 00/00/0000 (dd/mm/yyyy)

Authorized Proc. group : P10

Extended license : \*NONE

Registration log

License DOTNET

Get Activation key parameters  
OK  
Check license validity  
Connection succeeded.  
Disconnecting from new licence.  
Ready.

Check existing

OK Cancel Apply

Below information must be the same as on the form received from AURA (Equivalent to EASYREG command).

- **License** : Enter license.
- **Special development license** : Select this option for development license.
- **Only used in development** : Select this option for a development license which will never be used by an application.
- **Activation key** : Enter key (10 characters).
- **Company name** : Enter company name.
- **Number of connections** : Enter connection(s) number.
- **Authorized partition ID** : 0 (default)
- **EASYCOM's version** : 3 (default).
- **EASYCOM's option** : 0 (default).
- **Expiration date** : Enter key the end date, in dd/mm/yyyy format
- **Authorized Proc. Group** : \* (default).
- **Extended license** : \*NONE (default).

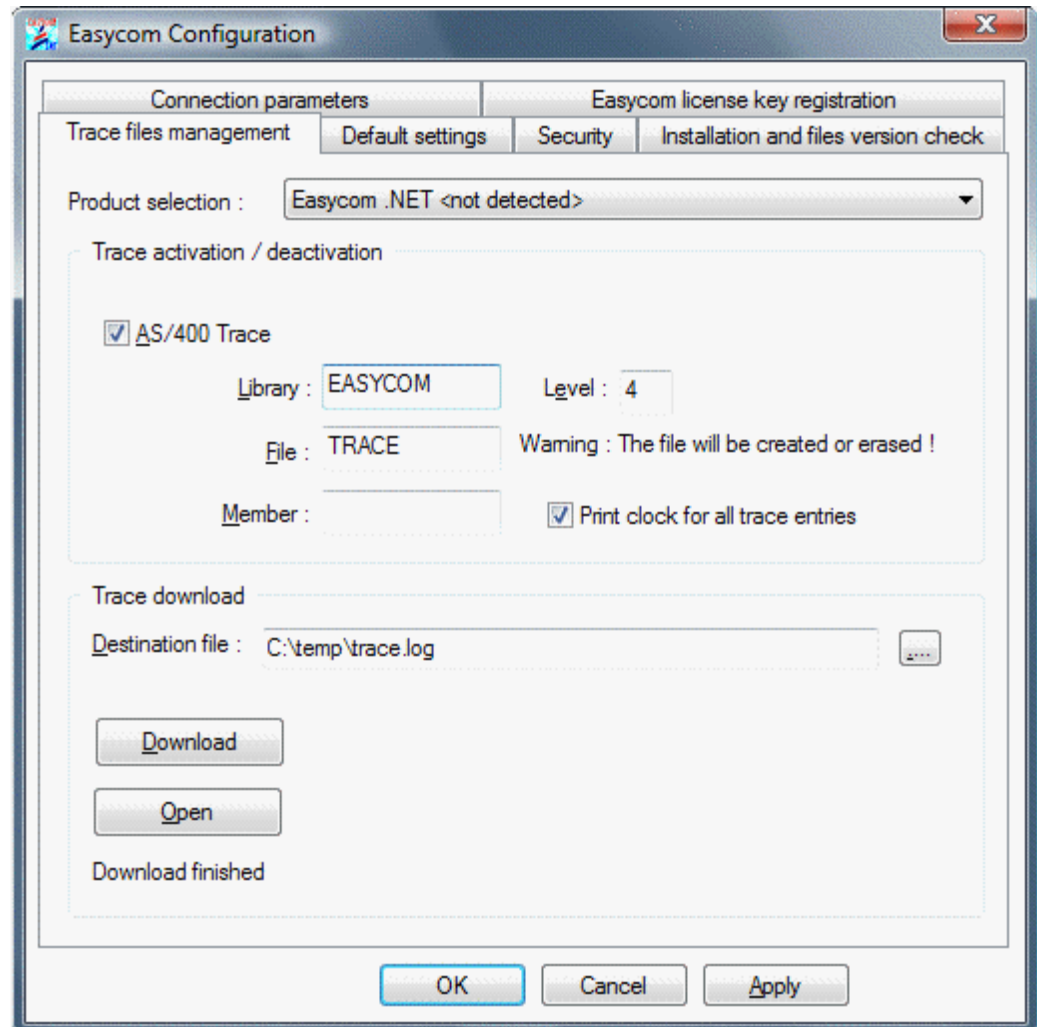
Press '**Register**' button to submit the registration process to the iSeries. After integrity check it will store it into the iSeries and test the connection on that license.

### Trace file management

In the event of an error or to audit EASYCOM operations, it is interesting to keep traces of what program performs.

AS/400 trace mode is devoted to this job.

Traces mode reduced performances significantly, it must be strictly reserved for analysis purpose.



### *Trace Activation / Deactivation*

- **AS/400 trace :**

Select "**AS/400 Trace**" to activate or deactivate the EASYCOM generated trace on AS/400.

- **Library :**

Use of AS/400 trace requires to specify at least an AS/400 library and file names.

Warning, library name must be one with writing rights opened.

- **File :**

The file will be created if it does not exist and deleted later. Commands to AS/400 can detail if suitable.

- **Level :**

Low detail level is 1 (default value), highest is 9. Trace level 4 is usually sufficient. At this level, all fields' values sent or received are detailed.

- **Operation time printing :**

Operation time printing gives an idea of elapsed time between each request. Level 1 is enough in this case.

- **Member :**

Optional option.

### Trace download

To download AS/400 generated trace, information related to AS/400 trace access is required. If AS/400 trace is already active, this information is already available.

- **Destination file :**

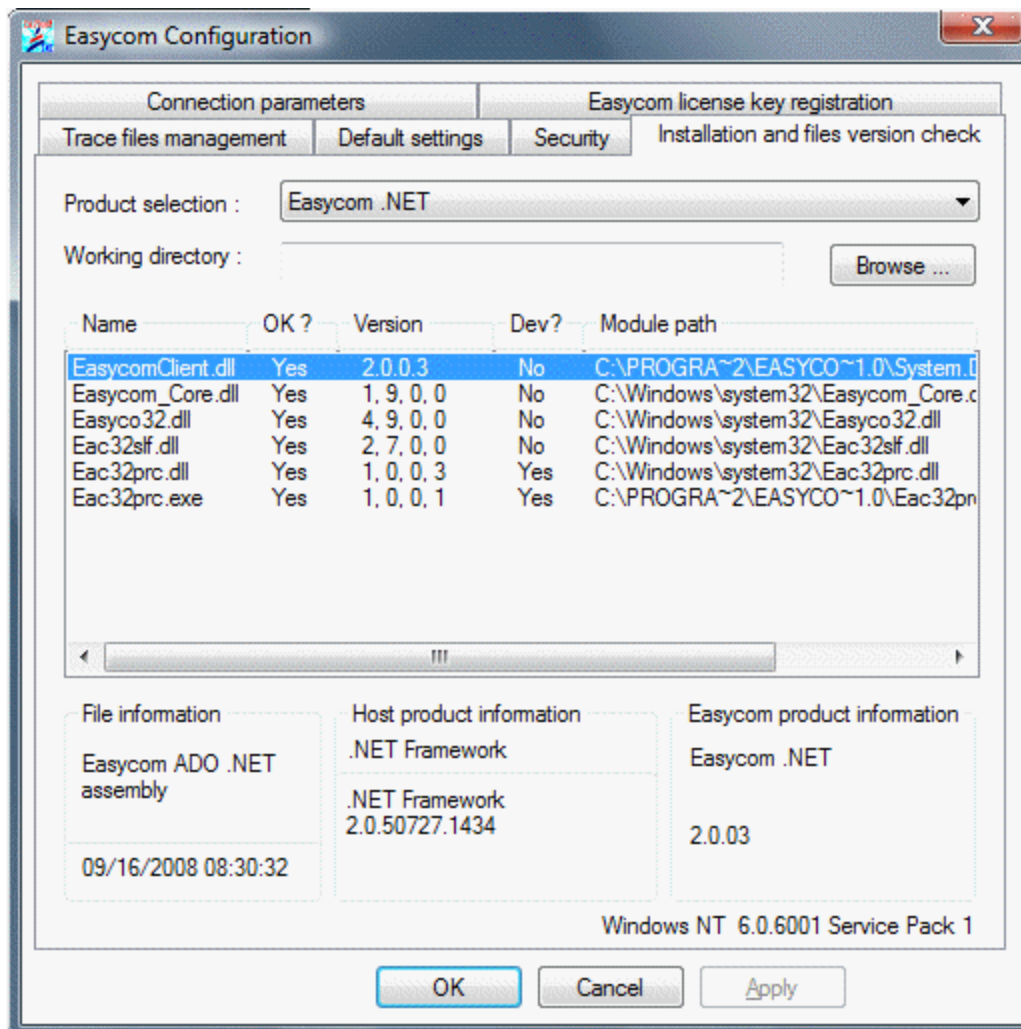
PC file should be specified, it will be generated by filling the entry box or choosing the file in the tree structure using the "**browse**" button.

NB: An user name and password must be specified on "Connection parameters" bookmark.

- **To download :**

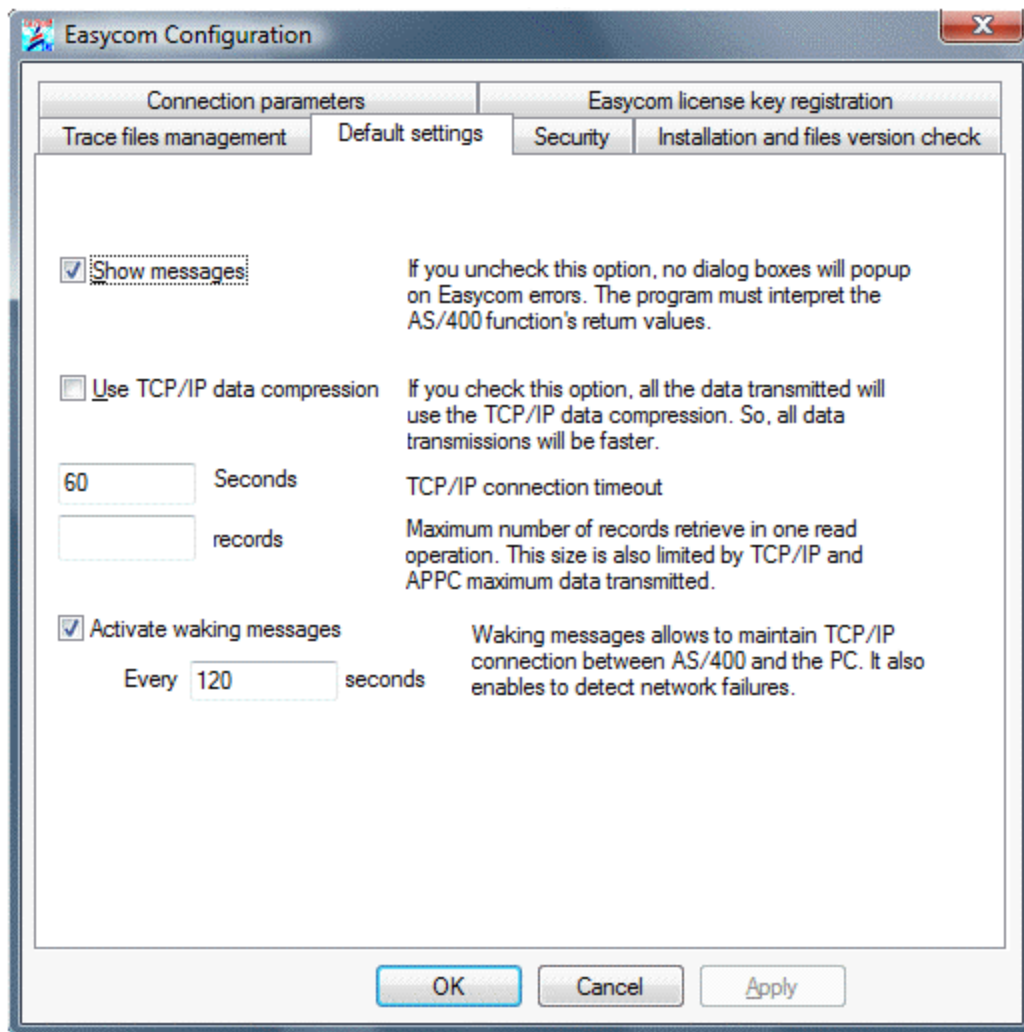
Click on "**Downloading**" to recover the trace.

### Installation and checking modules versions (DLLs)



### Easycom default settings

This part allows setting EASYCOM parameters to optimize network access times and reduce exchanges between AS/400 and application.



### Messages display

If this option is unchecked, no dialog box will be displayed in case of EASYCOM error. Then, program will interpret the functions returned values in all cases (example: password error). This option is recommended with a PC program server type (Web or any program operating automatically).

### TCP/IP data compression

This option allows to use data compression to reduce exchanged volumes between AS/400 and PC.

### TCP/IP connection maximum timeout

Default: 60 seconds.

Timeout = "": Default value 60s

Timeout = 0: no timeout

### Retrieved recordings maximum number

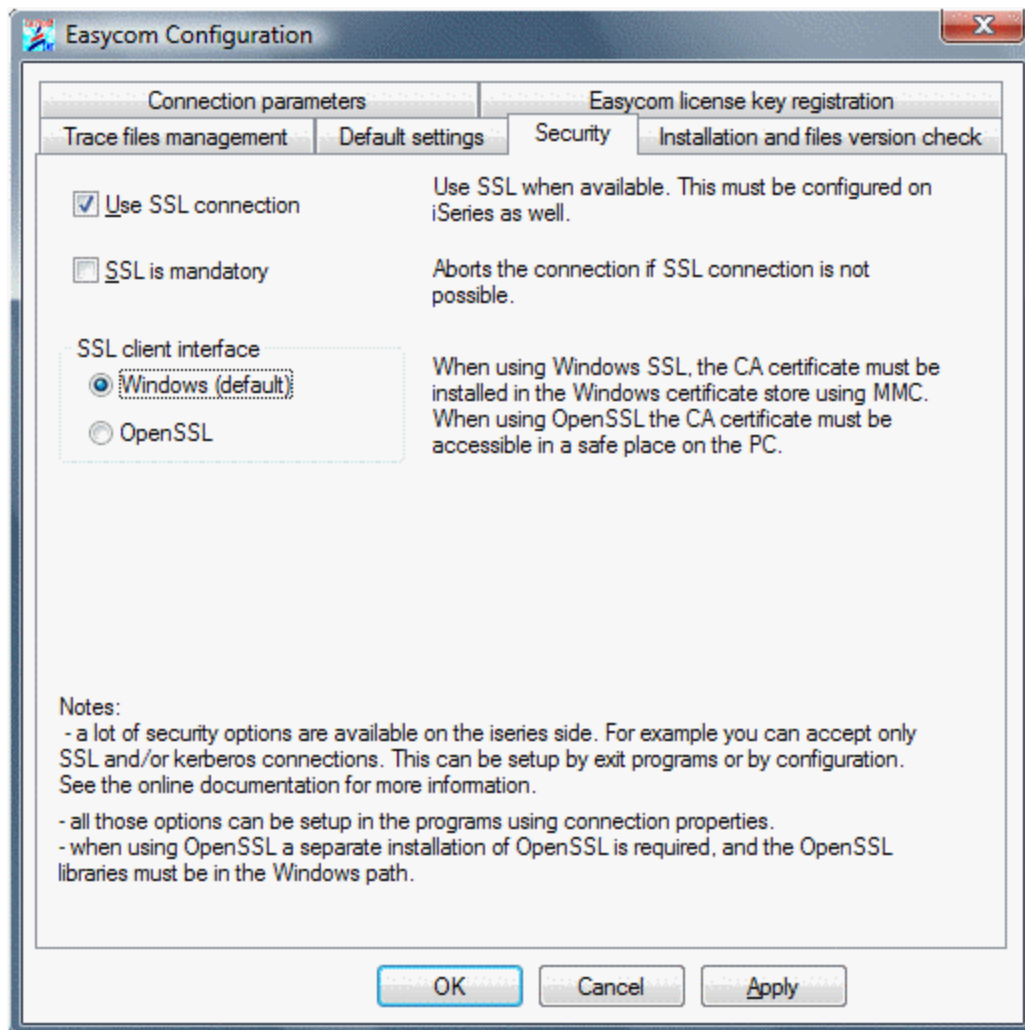
With this option maximum recordings number read in a block can be set. Default value is 32, and its limit is set by the block size parameter.

### Activating Keep Alive messages

In the case of sustained applications uses without data exchanged, TCP/IP may close AS/400 to PC communication. To avoid that, regular intervals messages can be sent on established connection. That also allows an EASYCOM job automatic termination in case of PC prolonged silence.

### Easycom security

This tab is for SSL default connectivity options. SSL connection settings can also be setup inside the client application.



If SSL is activated, SSL connection will be attempted. If the SSL negotiation fails or not supported by the server or client, the connection will continue without SSL (not encrypted).

If SSL is activated and mandatory, the client will successfully connect only if SSL negotiation succeeds.

Notes :

- if the client part is not up to date, the option may be ignored, and connection succeed without SSL
- if client part is up to date, but not server part, the connection will be aborted.
- The connection test tab shows if the connection test was successfully using SSL (yes, no or N/A for not supported on client). If this test succeeds, this does not mean that

the application will use SSL, because client part is specific for each product (Delphi, WinDev, PHP ...).

This configuration screen shows two different interfaces:

- Windows (default). Use the Microsoft Windows integrated interface. You may be need to install the certificate of the CA (certificate Authority) that issued the certificate of the SSL Easycom server ([see SSL connection - server configuration](#)).

To do this, use mmc (Microsoft Management console), and add the certificate store plugin into it. You can do this by clicking "start", "Run", and type "certmgr.msc" then enter. Then right-click on "Trusted Root Certification Authorities", then select "All Tasks", and the "Import". You need then to select the file that is containing the certificate.

- OpenSSL. Use OpenSSL interface. In this case the OpenSSL libraries must be available on the PC. You also need to have the CA certificate available. You can give the certificate path or name using Easycom configuration tool (or inside the application).

### Easycom.ini

The easycom.ini file contains parameters and comprehensive options (installation, optimization, trace, etc...) set including EASYCOM Configuration utility chosen parameters.

Several easycom.ini are possible. In this case it will be looked for first in the application repertory, then in the Windows repertory and finally in another path.

### Example : Easycom.ini file

```
[INSTALL]
PCdir=C:\PROGRAM FILES\Easycom

[GENERAL]
Network=
Msg=1 //Option 'Display messages'
NoWait=
QryOptimize=
Location=194.206.165.100 //AS/400 name or IP address

[TCP]
COMPRESSION=
Timeout=5 //TCP/IP connection maximum time

[Buffers]
Record=9 //recordings maximum Number
//retrieved in a reading operation
Size=8000 //data maximum size in byte
//sent between AS/400 and PC
TimeOut=30 //data refreshing time
```

## Use example : EacXML class

## Develop an application Windows Form

### Prerequisites

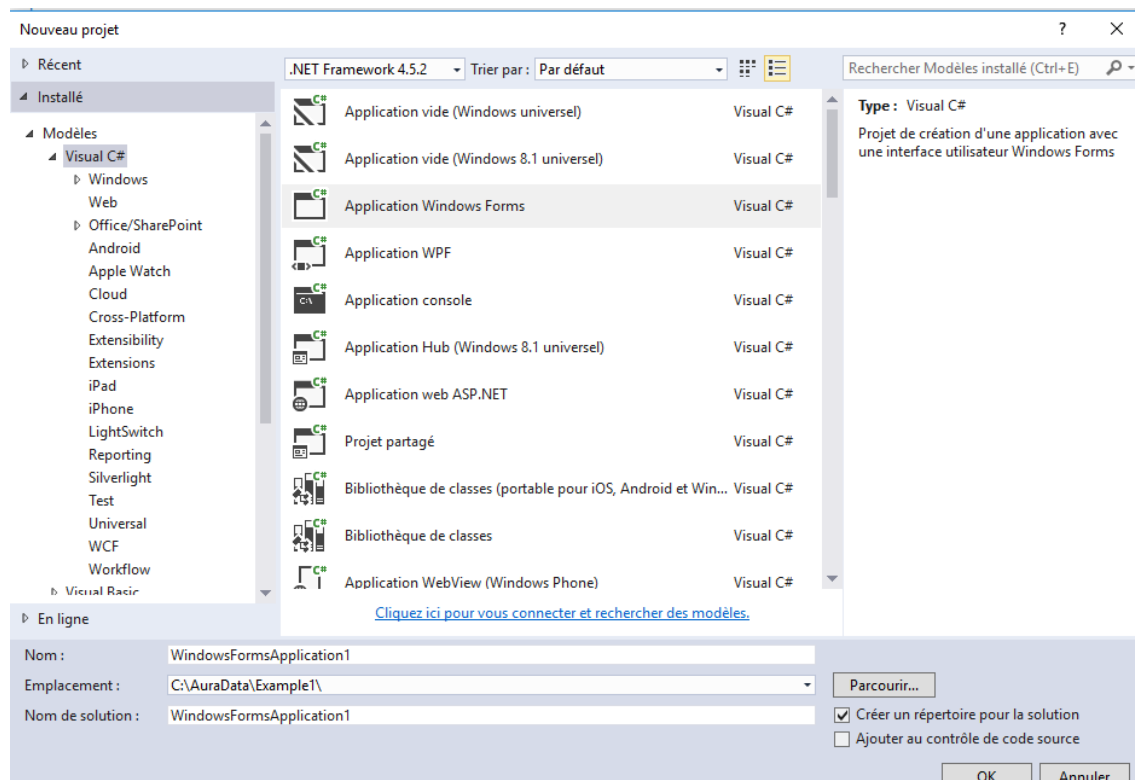
Easycom for .NET has been installed on the Windows PC on which the application will be developed with Visual Studio.

### Project setup

This example is available into the installation directory:

C:\Users\Public\Documents\Easycom For .NET Samples\AppliDemoXml.

In this example, we will create an Application Windows Form with Visual Studio :



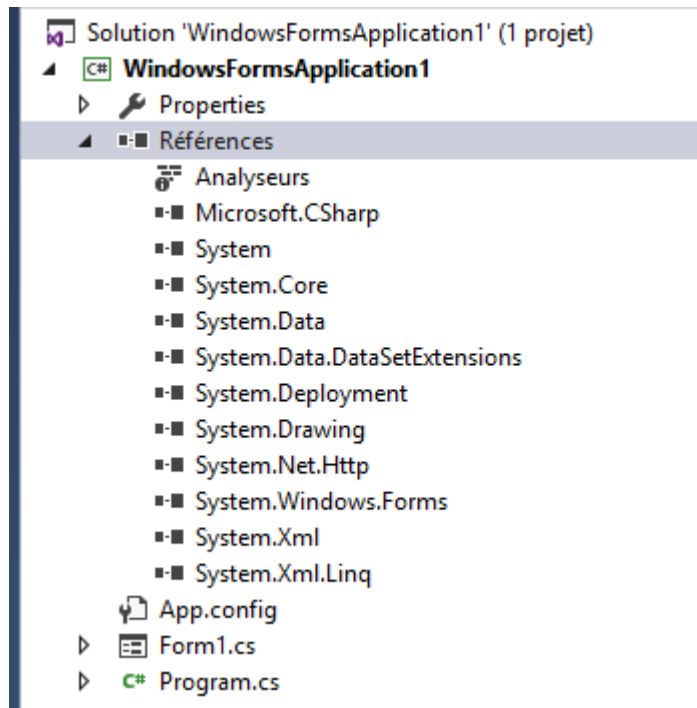
The .NET Framework target is the 4.6.1.

### Add Easycom for .NET reference

The Easycom for .NET assembly is located into the installation directory:

C:\Program Files (x86)\Easycom .NET\ System.Data.EasycomClient.dll

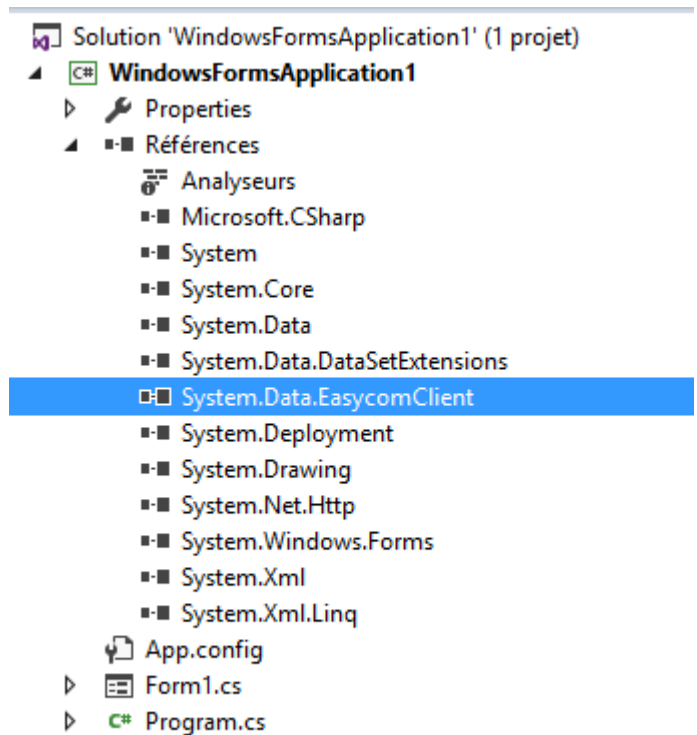
Into the project properties, right click on References and add a reference:



Browse directories to indicate this path:

C:\Program Files (x86)\Easycom .NET\System.Data.EasycomClient.dll

Then add the reference:



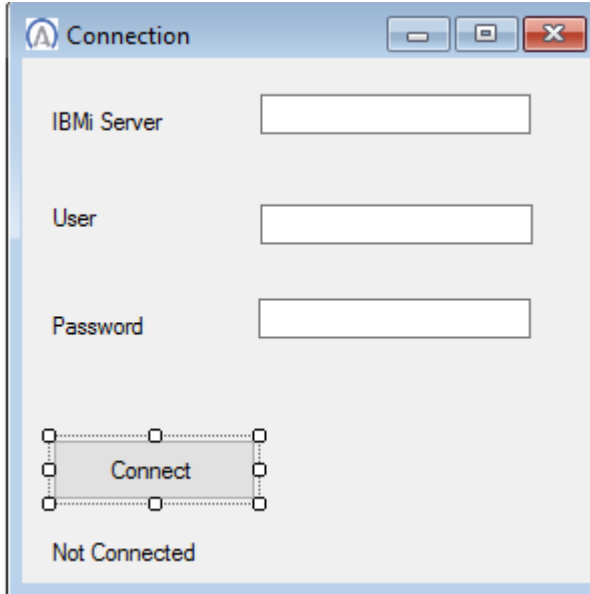
Now, into the Program, you can add the directive:

`using Easycom;`



## Open a connection with AS/400

In this example, we use a first Form to manage the connection with the AS400 machine:



To open a connection, we use the following class:

Easycom.[EasycomConnection](#)

and we define the following property:

```
public static EasycomConnection Connection;
```

A click on the "Connect" button will cause the following treatment:

```
if (textBoxServer.Text != "" && textBoxUser.Text != "" &&
    textBoxPwd.Text != "")
{
    Connection = new EasycomConnection();

    Connection.ConnectionString = "Server=" + textBoxServer.Text
    + ";User Id=" + textBoxUser.Text + ";Password=" +
    textBoxPwd.Text + ";Pooled=False";

    try
    {
        Connection.Open();
    }
    catch (Exception exp)
    {
        MessageBox.Show("Connection failed:" + exp.Message);
        Cursor.Current = Cursors.Default;
    }
}
```

## Call AS/400 program

Into this example we will call a program by using the following method : [EacXML.XMLExecRequest](#), available into the namespace Easycom.  
We call the Open List of Authorized Users (**QGYOLAUS**) IBM API. It provides information about the authorized users of the system.

Assuming we have this XML file(**Easycom\_Xml\_Define\_In.xml**):

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?><Easycom LogFile="/tmp/xml_demo_cedrick.log"
DebugLog="259">
  <define>
    <file type="file" stmf="stmf" />
  </define>
</Easycom>
```

And assuming we have this other XML file (**Easycom\_Xml\_Program\_In.xml**):

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?><Easycom LogFile="/tmp/xml_demo_cedrick.log"
DebugLog="259">
  <Program Name="name">
    <parameterList></parameterList>
  </Program>
</Easycom>
```

These files are present into the following directory:

**AppliDemoXml\NewAppliDemoXml\bin\Release**

By using the **System.Xml.Linq** classes, it is pretty easy to modify XML files and call program:

```
//////////////////////////////////// 1) Load program prototypes
String Type = "PCML";
String stmf1 = "/usr/local/easycom/PCML/QGYOLAUS.pcm1";
doc = XDocument.Load("Easycom_Xml_Define_In.xml");
Elmt = doc.Root.Element("define").Element("file");
Elmt.Attribute("type").Value = Type;
Elmt.Attribute("stmf").Value = stmf1;

stringXMLOut = myEacXML.XMLExecRequest(XMLToString(doc));

//////////////////////////////////// 2) Call QGYOLAUS system API
doc = XDocument.Load("Easycom_Xml_Program_In.xml");
Elmt = doc.Root.Element("Program");
Elmt.Attribute("Name").Value = "QGYOLAUS";

stringXMLOut = myEacXML.XMLExecRequest(XMLToString(doc));
```

This is the first request sent to the EASYCOM server to load prototype of QGYOLAUS and the EASYCOM server response:

REQUEST:  
<?xml version="1.0" encoding="utf-8" standalone="yes"?><Easycom LogFile="/tmp/xml\_demo\_cedrick.log"
DebugLog="259">
 <define>
 <file type="PCML" stmf="/usr/local/easycom/PCML/QGYOLAUS.pcm1" />
 </define>
</Easycom>



-----

RESPONSE:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Easycom Version="2.01"/>
```

This is the second request sent to the EASYCOM server to call QGYOLAUS and the EASYCOM server response :

=====

REQUEST:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?><Easycom LogFile="/tmp/xml_demo_cedrick.log"
DebugLog="259">
  <Program Name="QGYOLAUS">
    <parameterList></parameterList>
  </Program>
</Easycom>
```

-----

RESPONSE:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Easycom Version="2.01">
  <Program Name="qgyolaus">
    <ParameterList>
      <receiver Type="Array" count="0"/>
      <listInfo Type="Struct" Struct="listInfo">
        <totalRcds Type="Int">106</totalRcds>
        <rcdsReturned Type="Int">0</rcdsReturned>
        <rqsHandle Type="Byte" Encoding="BASE64">AAAAAQ==</rqsHandle>
        <rcdLength Type="Int">62</rcdLength>
        <infoComplete Type="Char">C</infoComplete>
        <dateCreated Type="Char">1160707</dateCreated>
        <timeCreated Type="Char">110642</timeCreated>
        <listStatus Type="Char">2</listStatus>
        <lengthOfInfo Type="Int">0</lengthOfInfo>
        <firstRecord Type="Int">0</firstRecord>
      </listInfo>
    </ParameterList>
  </Program>
</Easycom>
```

=====

If after we use the Get List Entries (**QGYGTLE**) IBM API, we will be able to get entries from previously opened Authorized Users list.

REQUEST:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?><Easycom LogFile="/tmp/xml_demo_cedrick.log"
DebugLog="259">
  <Program Name="qgygtle">
    <parameterList>
      <receiverLength>6572</receiverLength>
      <requestHandle>AAAAAQ==</requestHandle>
      <rcdsToReturn>106</rcdsToReturn>
      <STARTINGRCD>1</STARTINGRCD>
    </parameterList>
  </Program>
</Easycom>
```

-----

RESPONSE:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Easycom Version="2.01">
```

```
<Program Name="qgygtle">
  <ParameterList>
    <receiver Type="Array" count="106">
      <item Type="Struct" Struct="autu0150">
        <NAME Type="Char">AAABBBCCCD</NAME>
        <userOrGroup Type="Char">0</userOrGroup>
        <groupMembers Type="Char">0</groupMembers>
        <description Type="Char"/>
      </item>
      <item Type="Struct" Struct="autu0150">
        <NAME Type="Char">ADMIN</NAME>
        <userOrGroup Type="Char">0</userOrGroup>
        <groupMembers Type="Char">0</groupMembers>
        <description Type="Char">Responsable de la sécurité</description>
      </item>
      <item Type="Struct" Struct="autu0150">
        <NAME Type="Char">ZS5250DEMO</NAME>
        <userOrGroup Type="Char">0</userOrGroup>
        <groupMembers Type="Char">0</groupMembers>
        <description Type="Char">Zend 5250 demo user</description>
      </item>
    </receiver>
    <listInfo Type="Struct" Struct="listInfo">
      <totalRcds Type="Int">106</totalRcds>
      <rcdsReturned Type="Int">106</rcdsReturned>
      <rqshandle Type="Byte" Encoding="BASE64">AAAAAQ==</rqshandle>
      <rcdLength Type="Int">62</rcdLength>
      <infoComplete Type="Char">C</infoComplete>
      <dateCreated Type="Char">1160707</dateCreated>
      <timeCreated Type="Char">110642</timeCreated>
      <listStatus Type="Char">2</listStatus>
      <lengthOfInfo Type="Int">6572</lengthOfInfo>
      <firstRecord Type="Int">1</firstRecord>
    </listInfo>
  </ParameterList>
</Program>

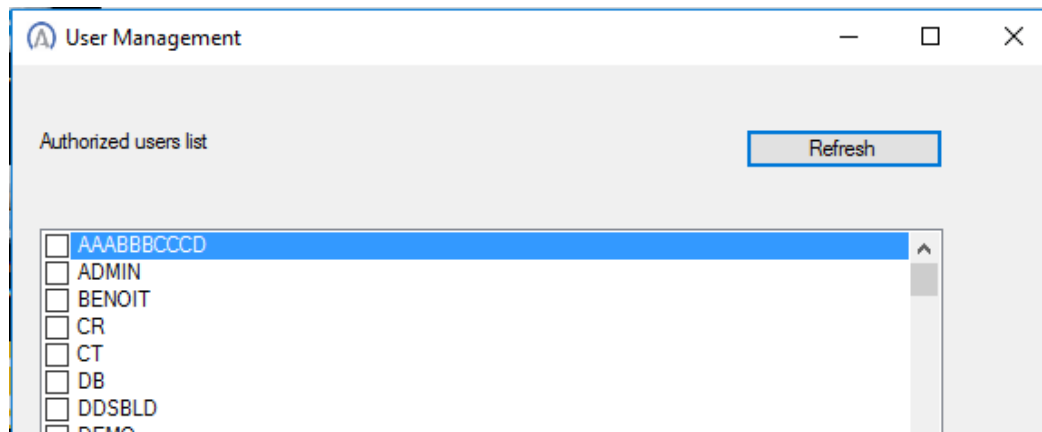
</Easycom>
=====
```

By using the **System.Xml.Linq** classes, it is also pretty easy to parse XML out and retrieve users names (**Program.ParameterList.receiver.item.NAME elements in red above**):

```
XElement root2 = XElement.Parse(stringXMLOut);
List<string> userList = new List<string>();

IEnumerable<XElement> item2 =
    from el in
        root2.Element("Program").Element("ParameterList").Element("receiver").Elements("item")
    select el.Element("NAME");
foreach (XElement el in item2)
{
    userList.Add(el.Value);
}

BindingSource bs = new BindingSource();
bs.DataSource = userList;
checkedListBoxUsers.DataSource = bs;
```



## Deploy the developped Windows Form application

### Prerequisites

The target platform has to have installed at least the target .NET Framework.

### Deployment

The target system has the following features:

- 64-bit Windows 10
- .NET Framework

The files to deploy are:

- C:\Program Files (x86)\Easycom .NET\System.Data.EasycomClient.dll : the main assembly
- C:\Program Files (x86)\Easycom .NET\x64\Easycom\_core.dll : the x64 core file
- The application executable: appli.exe
- XML files used into the application: Easycom\_Xml\_Define\_In.xml and Easycom\_Xml\_Program\_In.xml

## Use example: EasycomConnection class and RemoteRtvCommand method

This example is available into the installation directory:

C:\Users\Public\Documents\Easycom For .NET Samples\EasycomVB.

It has been developed in VB with Visual Studio.

Into this exemple, we use the Retrieve Job Attributes (RTVJOBA) command, to retrieve the user profile associated with the EASYCOM job created by the connection.

```
Imports System.Data.EasycomClient

Dim my_cnx As EasycomConnection

my_cnx = New EasycomConnection
my_cnx.ConnectionString = "Server=power8;User Id=qpgmr;Password=launcher"
my_cnx.Open()
my_cnx.RemoteRtvCommand("LIBL=CHAR(2750);DFTWAIT=DEC(7 0);RTVJOBA
USRLIBL(&LIBL)")
If my_cnx.RemoteRtvCommandGetValue("RC") = "0" Then
    MessageBox.Show("User Library List: " +
my_cnx.RemoteRtvCommandGetValue("LIBL"))
    MessageBox.Show("User Default Wait: " +
my_cnx.RemoteRtvCommandGetValue("DFTWAIT"))
Else
    MessageBox.Show("The command call returned the following message" +
my_cnx.RemoteRtvCommandGetValue("RC"))
End If

my_cnx.Close()
```

## Use example: EacConnection and EacCommand classes

This example is available into the installation directory:

C:\Users\Public\Documents\Easycom For .NET Samples\EasycomVB.

It has been developed in VB with Visual Studio.

Into this exemple, we execute the EASYCOMXMP/RPCSAMPLE program.

This program has been first described with the [RPC/Data Queue Configurator](#).

```
Imports System.Data.EasycomClient

Dim res As Int32
Using easyConn As System.Data.EasycomClient.EacConnection = New
System.Data.EasycomClient.EacConnection()
    easyConn.ConnectionString = "Server=power8;User Id=;Password=;Init
Libl=easycomxmp"
    easyConn.Open()

    Using easyComm As System.Data.EasycomClient.EacCommand = New
System.Data.EasycomClient.EacCommand()

        With easyComm

            .Connection = easyConn
            .CommandType = CommandType.StoredProcedure
            .CommandText = "*PGM/RPCSAMPLE"

            With .Parameters.Add("OP1", DbType.Double, 1)
                .Direction = ParameterDirection.Input
                .Value = OP1.Text
            End With

        End With

    End Using

End Using
```



```
With .Parameters.Add("STR1", DbType.StringFixedLength, 1)
    .Direction = ParameterDirection.Input
    .Value = STR1.Text
End With
With .Parameters.Add("OP2", DbType.Double, 1)
    .Direction = ParameterDirection.InputOutput
    .Value = OP2.Text
End With
With .Parameters.Add("STR2", DbType.StringFixedLength, 1)
    .Direction = ParameterDirection.InputOutput
    .Value = STR2.Text
End With
With .Parameters.Add("OP3", DbType.Double, 1)
    .Direction = ParameterDirection.Output
End With

End With

res = easyComm.ExecuteNonQuery()

OP1.Text = easyComm.Parameters.Item(0).Value.ToString()
STR1.Text = easyComm.Parameters.Item(1).Value.ToString()
OP2.Text = easyComm.Parameters.Item(2).Value.ToString()
STR2.Text = easyComm.Parameters.Item(3).Value.ToString()
OP3.Text = easyComm.Parameters.Item(4).Value.ToString()

End Using
easyConn.Close()

End Using
```

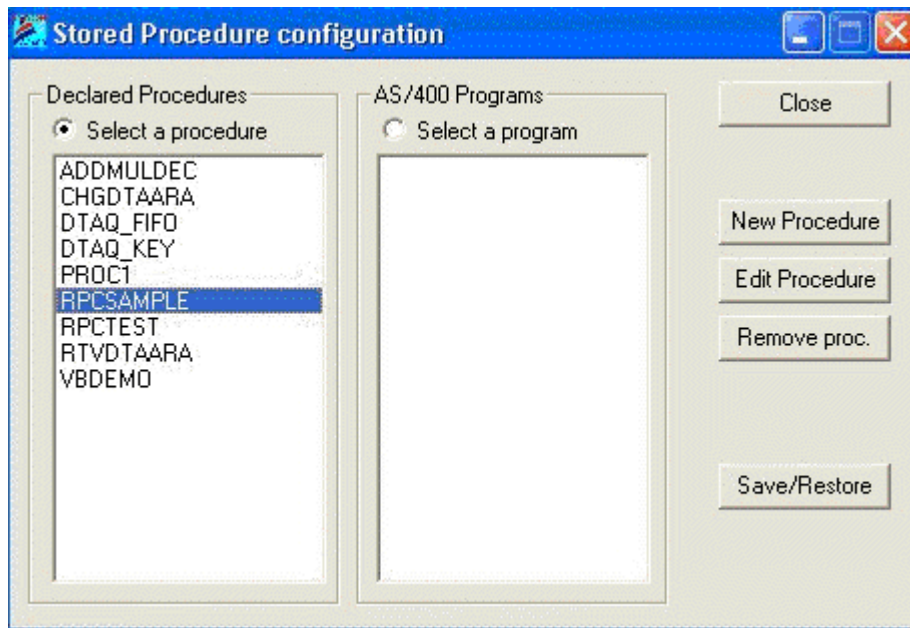
### AS/400 Native Programs Description

EASYCOM enables AS/400 native programs calling, CL or RPG programs or stored procedures.

To perform this, EASYCOM needs these programs description stored on AS/400 in YPROCHDR and YPROCPARMS files in EASYCOM library.

Programs description and data queues are built by DTAQ-RPC constructor. The basic principle is to specify all parameters, types and uses (input, output, input/output) required to call the program.

The first screen displays the existing procedures (stored on AS/400) and enables to create, modify, or delete them. The descriptions can be saved in a PC text file in view of a later transfer to another AS/400.



A new name is assigned to the procedure. It does not need to match with the associated program name.

A native AS/400 program (CL, RPG, COBOL, C etc.) is associated to the procedure.

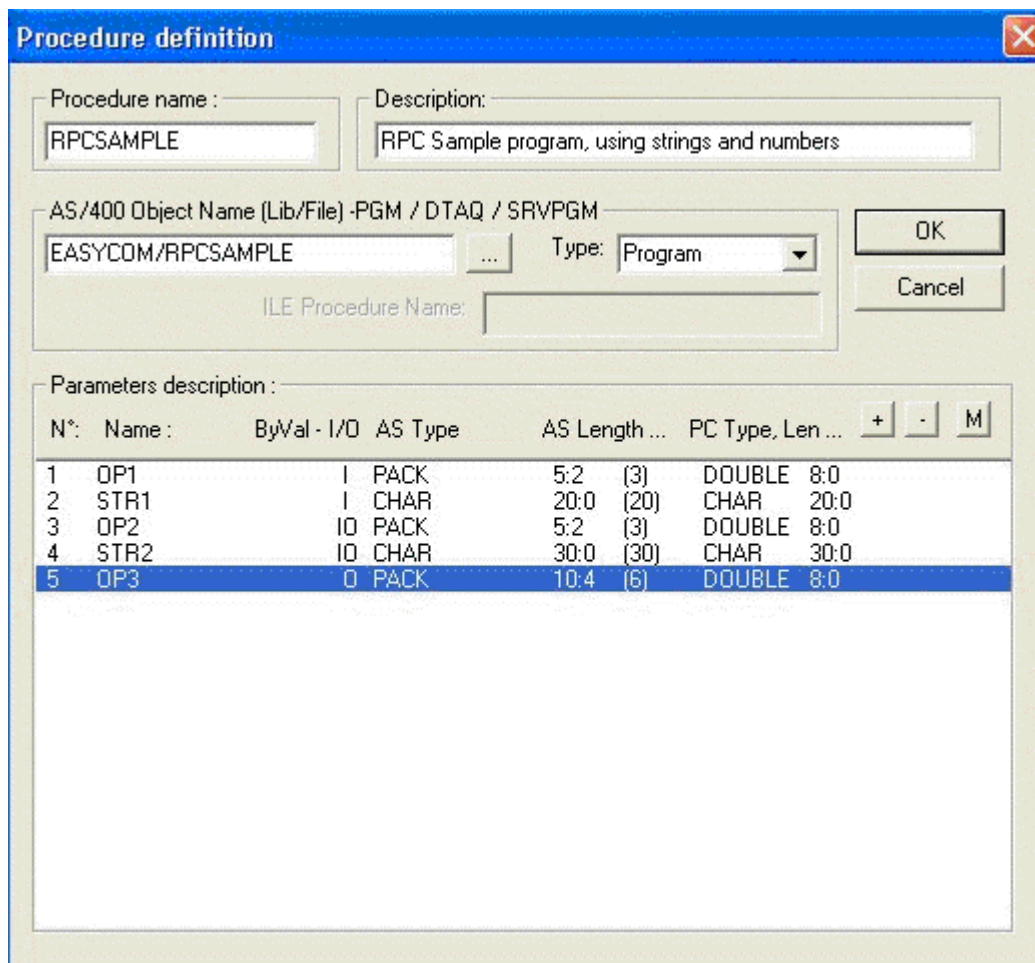
The library may be omitted or replaced by \*LIBL.

The description is a free text, which will be seen when client workstations browse through the procedures.

Each program calling type and size parameters are described.

Each parameter may be considered as a database table field.





**Procedure definition**

Procedure name :  Description:

AS/400 Object Name (Lib/File) -PGM / DTAQ / SRVPGM:  ... Type:

ILE Procedure Name:

OK Cancel

Parameters description :

N°	Name	ByVal	I/O	AS Type	AS Length	PC Type	Len	
1	OP1		I	PACK	5:2 (3)	DOUBLE	8:0	
2	STR1		I	CHAR	20:0 (20)	CHAR	20:0	
3	OP2		IO	PACK	5:2 (3)	DOUBLE	8:0	
4	STR2		IO	CHAR	30:0 (30)	CHAR	30:0	
5	OP3		O	PACK	10:4 (6)	DOUBLE	8:0	

+ - M

Each parameter can be considered as a field in a database table.

It therefore has a name, by which it can be referred to by the application.

Parameters designed to provide values for the called program are considered as input parameters (IN).

Parameters designed to receive a value on returning from the call are considered as output parameters (OUT).

Parameters that are modified by the program are both input and output (IN/OUT) parameters.

By default, all the parameters in an AS/400 program are both input and output. The logic of the program can change this property.

If a calling parameter of the program is a structure (DS: Data Structure), each field of the DS has to be described individually.

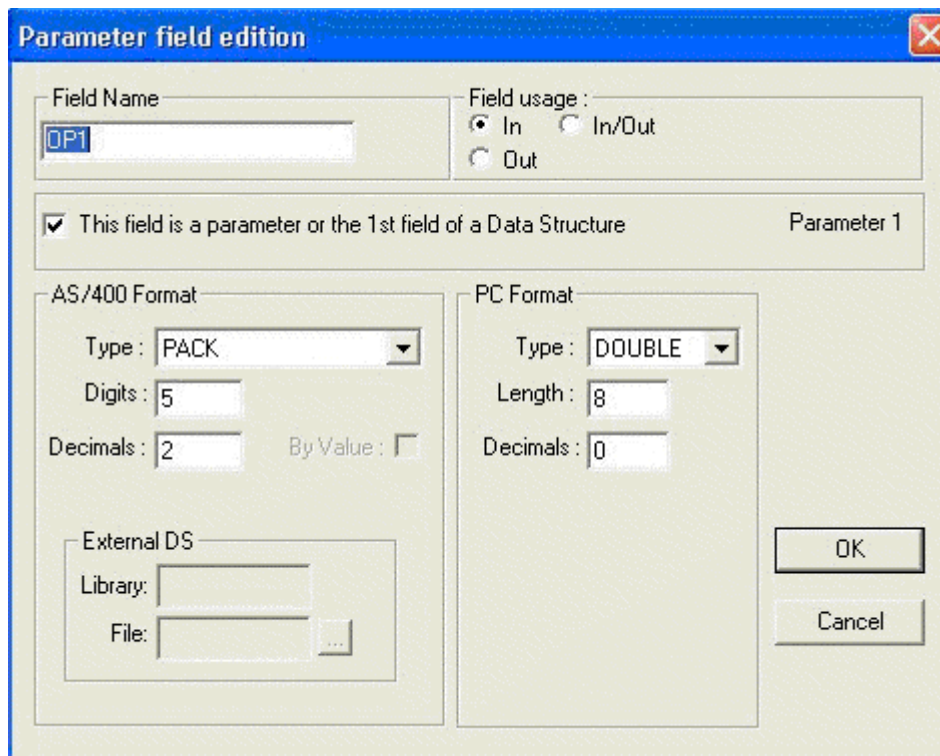
For the first field only, the box to be ticked is: This field is a parameter or the 1st Field.

The type of parameter expected by the AS/400 program must be specified exactly :

CHAR : Character data type.  
 BIN2 : 16-bit numeric data type.  
 BIN4 : 32-bit numeric data type.  
 PACK : Condensed numeric data type (DECIMAL).

This is the format in which CL handles numerical data (CL \*DEC type).

ZONED : Extended numeric data type (NUMERIC).  
 DATE : AS/400 date in the yyyy-mm-dd format.  
 TIME : Time in hh:mm:ss format.  
 FLOAT : Numeric value in single-precision floating point.  
 DOUBLE : Numeric value in double-precision floating point.  
 TIMESTP : Elapsed time field.  
 GRAPHIC : Character type data, not to be converted.  
 EXTERNAL DS : A structure described by an external data structure, i.e. a physical file.



**Parameter field edition**

Field Name:

Field usage: ☒ In ☐ In/Out ☐ Out

☒ This field is a parameter or the 1st field of a Data Structure Parameter 1

AS/400 Format

Type:

Digits:

Decimals:  ☐ By Value

PC Format

Type:

Length:

Decimals:

External DS

Library:

File:

OK Cancel

## Other use examples

Some other examples are also available into the installation directory:

C:\Users\Public\Documents\Easycom For .NET Samples

All have been developed in VB with Visual Studio.

### 1) Easycom Sample:

#### “Program call”:

- EacConnection class
- RemoteCommand method

#### “Get information”:

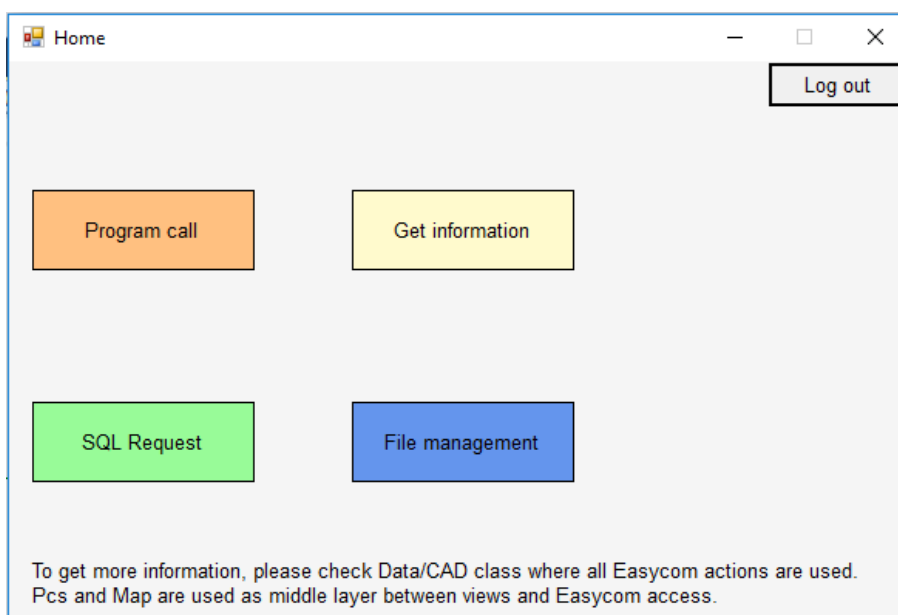
- EacConnection class:
  - RemoteRtvCommand method
  - RemoteRtvCommandGetValue method

#### “SQL request”:

- EacCommand class
  - ExecuteReader method
- EacDataReader class

#### “File management”:

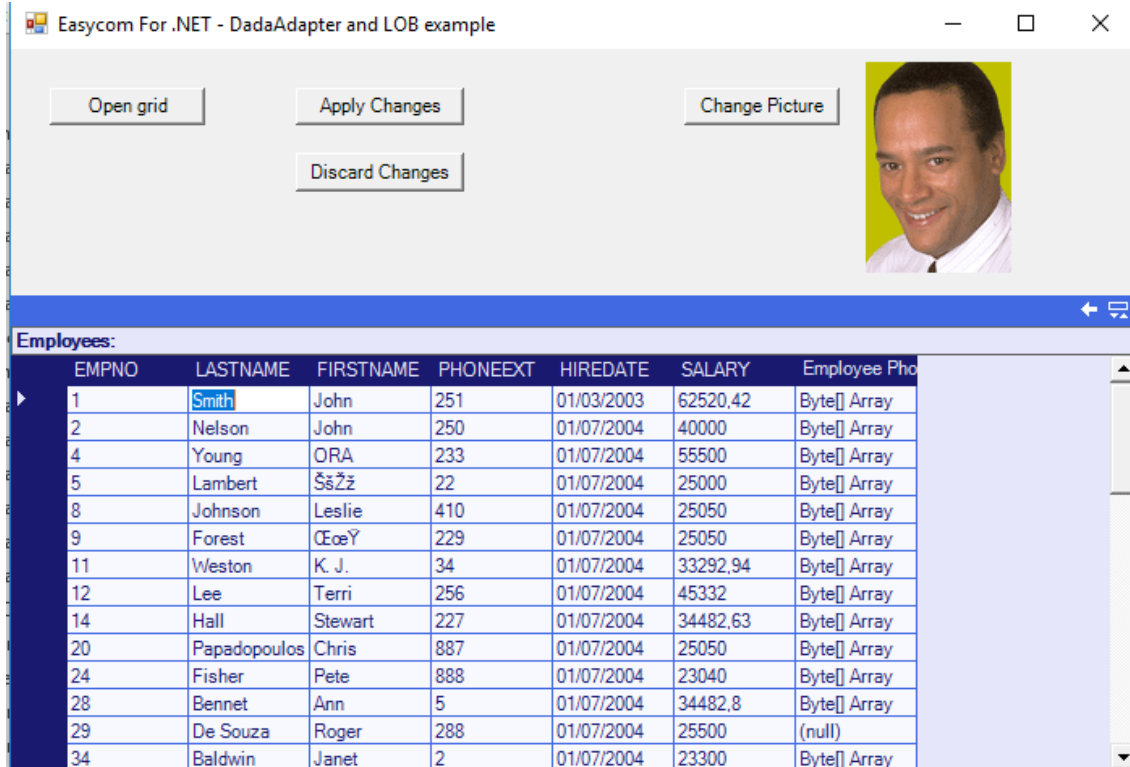
- EasycomFile class



## 2) Easycom\_DataAdapter

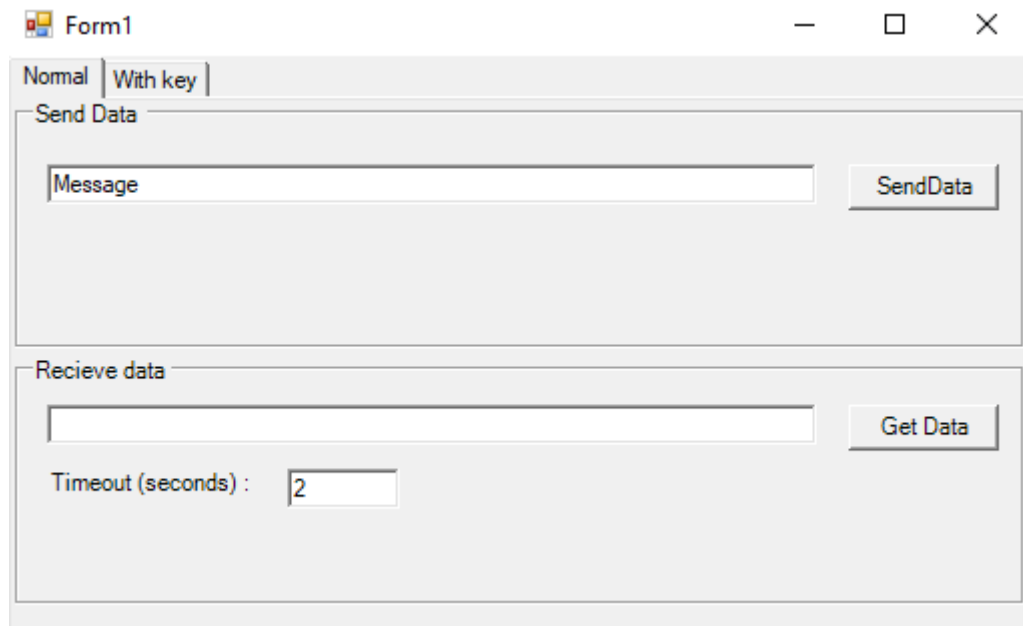
Not usable with .NET8, because this example uses System.Windows.Forms.DataGrid.  
System.Windows.Forms is not available with .NET8

- EacCommand class
- EacDataAdapter class



## 3) Easycom\_DataQueue

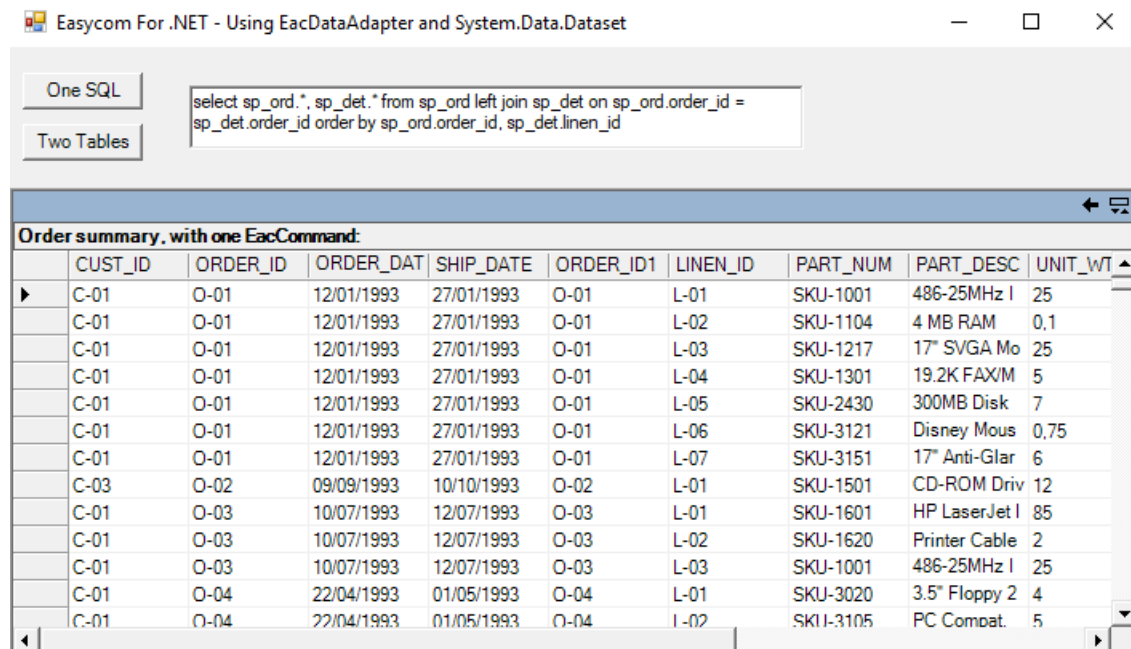
- EasycomFile class
  - RecordAppend method
  - SetValue method
  - RecordUpdate method
  - ReadKey method



#### 4) Easycom\_DataSet

Not usable with .NET8, because this example uses System.Windows.Forms.DataGrid.  
System.Windows.Forms is not available with .NET8

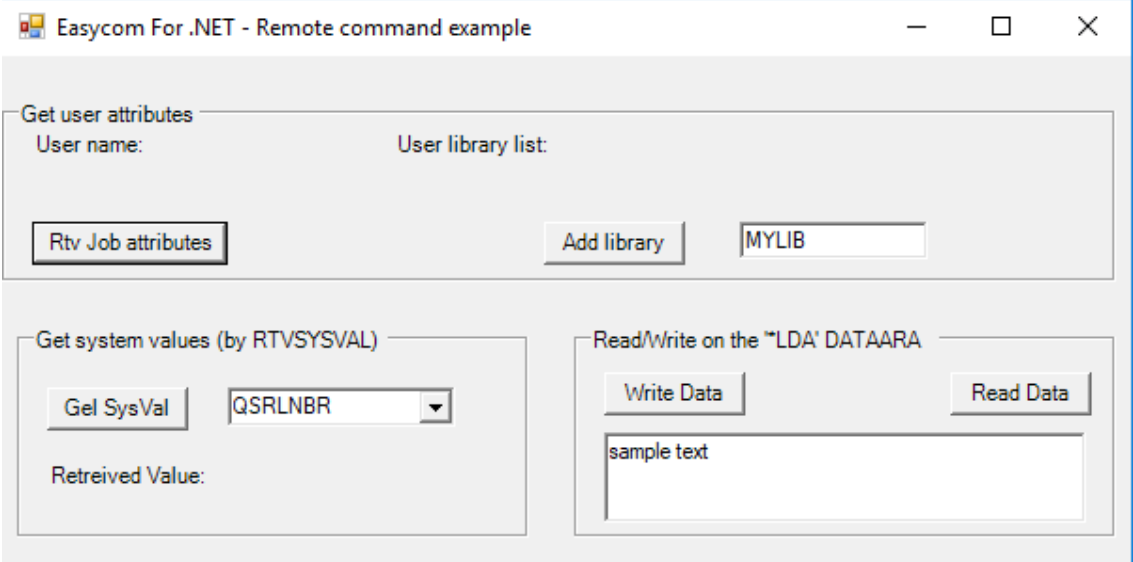
- EacCommand class
- EacDataAdapter class



	CUST_ID	ORDER_ID	ORDER_DAT	SHIP_DATE	ORDER_ID1	LINEN_ID	PART_NUM	PART_DESC	UNIT_WT
▶	C-01	O-01	12/01/1993	27/01/1993	O-01	L-01	SKU-1001	486-25MHz I	25
	C-01	O-01	12/01/1993	27/01/1993	O-01	L-02	SKU-1104	4 MB RAM	0,1
	C-01	O-01	12/01/1993	27/01/1993	O-01	L-03	SKU-1217	17" SVGA Mo	25
	C-01	O-01	12/01/1993	27/01/1993	O-01	L-04	SKU-1301	19.2K FAX/M	5
	C-01	O-01	12/01/1993	27/01/1993	O-01	L-05	SKU-2430	300MB Disk	7
	C-01	O-01	12/01/1993	27/01/1993	O-01	L-06	SKU-3121	Disney Mous	0,75
	C-01	O-01	12/01/1993	27/01/1993	O-01	L-07	SKU-3151	17" Anti-Glar	6
	C-03	O-02	09/09/1993	10/10/1993	O-02	L-01	SKU-1501	CD-ROM Driv	12
	C-01	O-03	10/07/1993	12/07/1993	O-03	L-01	SKU-1601	HP LaserJet I	85
	C-01	O-03	10/07/1993	12/07/1993	O-03	L-02	SKU-1620	Printer Cable	2
	C-01	O-03	10/07/1993	12/07/1993	O-03	L-03	SKU-1001	486-25MHz I	25
	C-01	O-04	22/04/1993	01/05/1993	O-04	L-01	SKU-3020	3.5" Floppy 2	4
	C-01	O-04	22/04/1993	01/05/1993	O-04	L-02	SKU-3105	PC Compat.	5

#### 5) Easycom\_RemoteCommand

- EacConnection class :
  - RemoteRtvCommand method
  - RemoteRtvCommandGetValue method
  - RemoteCommand method



## 6) Easycom\_StoredProcedure

Not usable with .NET8, because this example uses System.Windows.Forms.DataGrid.  
System.Windows.Forms is not available with .NET8

- EacCommand class
- EacDataAdapter class

Easycom For .NET - Calling a Stored Procedure that returns result sets and parame... — □ ×

Cust.ID:

Nbr of Orders:

Total paid:

Call S\_ORD\_SUM procedure