Link2i for .Net CORE

Access to Power i resources regardless of the operating system (Linux, Win32, Win64, OSX)

V6R1 - Beta

CONTENTS

Introduction

Link2iFor.Net Core is an API that allows developing information systems with .Net Core technology while benefiting from IBM i storage and / or background processing resources.

In addition to the API, Link2iFor.Net Core provides examples implementing and explaining all the functions.

The IBM i has experienced a major evolution of its data management system from the V5R4M0. Link2iFor.Net Core can't be used in older version. **S36 environment is not supported.** To work with that environment, you have to use CONNECT module developed in .Net Framework.

Installation of Link2iFor.Net Core

Link2i For.Net Core can reach all IBM i if the API Link2iForCloud (based on .Net Framework) is locally implemented to the IBM i targeted. In addition, we can use IBM Secure Gateway in that configuration.

To get IBM i benefits, Link2i For.Net Core consumes WCF web services provided by Link2iForCloud.

In that document, we use C# syntax and the reference to web services is done by the prefix “Client1.”. Only C# language can be used with Link2i For.Net Core.

General ratings

Idt must be an encrypted field that contains the IP address of the IBM i, a user ID, and a password.
An encrypted Id can be obtained using the following function:

Syntax:

```
{
public static string Idt, IP, Usr, Pwd;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.KryptAsync("[xx];" + IP + "|" + Usr + "|" + Pwd);
}
```

1 parameter composed with 4 separated values is required:

1. [xx] : contains a number between "01" and "31" ending by a ";"
2. IP.Text: IP address of the IBM I ending by a "|"
3. Usr.Text: contains the user ID ending by a "|"
4. Pwd.Text: contains the password of the user ID

The variable named "Idt" retrieves an encrypted identifier (String type). It must be used as it is in your further code.

Knowledge about targeted IBM i

IBM i OS name and version recovery as well as a session ID

NOSERI function

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.NOSERIAsync(Idt, SessionId, "LKWORK");
}
```

3 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. SessionId: contains the current HTML session ID (String type)
3. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

The returned result contains the system name, the OS version and processor group.

Databases inventory

ListDataBase function

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.ListDataBaseAsync(Idt, SessionId, "LKWORK");
}
```

3 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. SessionId: contains the current HTML session ID (String type)
3. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

This function returns a serialized XML stream providing databases list of an IBM i.

Tables inventory in a database

ListTables function

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.ListTablesAsync(Idt, Base, SessionId, "LKWORK");
}
```

4 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. Base: contains the name of the database for which the tables are listed (String type)
3. SessionId: contains the current HTML session ID (String type)
4. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

The result is a serialized XML stream providing tables list of a database of an IBM i.

Table fields list

ListField function

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await.Client1.ListFieldAsync(Idt, Base, Table, SessionId, Base2, Table2,
"LKWORK");
}
```

7 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. Base: contains the name of the database where the table is located (String type)
3. Table: contains the name of the table whose fields are to be listed (String type)
4. SessionId: contains the current HTML session ID (String type)
5. Base2: name of the database of the requested file
6. Table2: name of the requested file
7. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

The result is a serialized XML stream providing fields list of a file of a database of an IBM i. In addition, we get the name of the database, the name of the file and the number of records of the file.

Visualization of the detailed description of an object

ObjectDetail function

Syntax:

```
{
public static string XMLString;
}
```

```
XML_String = x.ObjectDetail(Idt, Base, Objet, ObjType, Session.SessionID, "LKWORK");
```

6 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. Base: contains the name of the database of the object to exam (String type)
3. Objet: contains the name of the object to exam (String type)
4. ObjType: contains the IBM i type of the object to exam (*PGM, *FILE, *DEVD, etc…) (String type)
5. Session.SessionID: contains the current HTML session ID (String type)
6. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

The result is a serialized XML stream providing the detailed description of an object.

Using IBM i programs

All programs encountered in IBM i are concerned by this part: all types of RPG, COBOL, CLP etc ……

Inventory of programs of a database

ListPgm function

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.ListPgmAsync(Idt, Base, SessionId, "LKWORK");
}
```

4 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. Base: contains the name of the database for which the programs are listed (String type)
3. SessionId: contains the current HTML session ID (String type)
4. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

The result is a serialized XML stream providing programs list of a database of an IBM i.

Inventory of source files of a database

ListSrc function

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.ListSrcAsync(Idt, Base, SessionId, "LKWORK");
}
```

4 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. Base: contains the name of the database for which the source files are listed (String type)
3. SessionId: contains the current HTML session ID (String type)
4. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

The result is a serialized XML stream providing source files list of a database of an IBM i.

Inventory of source members of a source file of a database

ListMbr function

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.ListMbrAsync(Idt, Base, SrcFile, SessionId, "LKWORK");
}
```

5 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. Base: contains the name of the database for which the members of source file are listed (String type)
3. SrcFile: contains the source file name for which the source members are listed (String type)
4. SessionId: contains the current HTML session ID (String type)

5. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

The result is a serialized XML stream providing source members list of a source file of a database of an IBM i.

Exporting a source member in an Excel Edit structure

SourceXLS function

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.SourceXLSAsync(Idt, Base, SrcFile, Mbr, Path, file, SessionId,
"LKWORK");
}
```

8 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. Base: contains the name of the database for which a source file member is exported (String type)
3. SrcFile: contains the name of the source file for which a source member is exported (String type)
4. Mbr: contains the name of the source member to export (String type)
5. Path: contains the name of the directory at the root of the server where the Excel file is placed (must end with a "/") (String type)
6. file: contains the name of the resulting Excel file (String type)
7. SessionId: contains the current HTML session ID (String type)
8. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

The result of this function is an Excel file that can then be imported for editing the source member, or the error message label if occurred.

Exporting a source member to a text file

SourceTXT function

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.SourceTXTAsync(Idt, Base, SrcFile, Mbr, Path, file, SessionId,
"LKWORK");
}
```

8 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. Base: contains the name of the database for which a source file member is exported (String type)
3. SrcFile: contains the name of the source file for which a source member is exported (String type)
4. Mbr: contains the name of the source member to export (String type)
5. Path: contains the directory name at the root of the server where the Text file is placed (String type)
6. file: contains the name of the Text file (String type)
7. SessionId: contains the current HTML session ID (String type)
8. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

The result of this function is a Text file or the error message label if occurred.
This export is useful for inventorying and analyzing the source codes present in an IBM i.

Generating an executable (compilation)

CrtPgm function

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.CrtPgmAsync(Idt, Base, SrcFile, Mbr, SessionId, "LKWORK",
DestBase, PgmName, SrcType);
}
```

9 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. Base: contains the name of the database for which a source file member is compiled (String type)
3. SrcFile: contains the name of the source file for which a source member is compiled (String type)
4. Mbr: contains the name of the source member to compile (String type)
5. SessionId: contains the current HTML session ID (String type)
6. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)
7. DestBase: contains the name of the destination database of the compiled program (String type)
8. PgmName: contains the name of the compiled program (String type)
9. SrcType: contains the type of source to compile (RPG, CBL, CLP etc…) (String type)

Associated with the previous function, this function can be useful for mass changes to provide to present programs in an IBM i.

Running an IBM i command

RUNCMD function

This function executes a CL command (ex: CRTLIB, CHGPF etc ...) on IBM i. The return is an XML string (here XMLString) and contains the message sent by IBM i at the end of execution (success or failure).

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.RUNCMDAsync(Idt, CMD);
}
```

This function runs an IBM I command (CRTLIB, the library creation command for example).

2 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. CMD: the command to execute (String type)

Running an existing program on IBM i

PROCESS_S function

This function runs a program developed on IBM i regardless of the language. Note that S36 is not supported.

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.PROCESS_SAsync(Idt, BIB, PGM, " ");
}
```

This function executes the program PGM of the BIB library and retrieves the message sent by the IBM i after the execution in the variable “ErrorMsg”.

4 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. BIB: the library that contains the program (String type)
3. PGM: the name of the program to launch (String type)
4. LOG: used only for working in progress (String type)

Running an existing program on IBM i with input parameters

PROCESS_i function

This function executes a program developed on the IBM i whatever the language. Note that S36 is not supported

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.PROCESS_iAsync("Idt", BIB, PGM, PRM, VALPRM, SEP, LOG);
}
```

This function executes the program PGM of the BIB library with parameter passing and retrieves the message sent by the IBM i after the execution in the return variable “ErrorMsg” (success or failure).

Parameters are sent to the program (PRM contains their respective values) with their respective lengths (in VALPRM) as expected in the program. Values and parameters are separated by the separator contained in the variable “SEP’’.

7 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. BIB: the library that contains the program (String type)
3. PGM: the name of the program to launch (String type)
4. PRM: the parameter(s) as a variable or a constant (String type)
5. VALPRM: the respective size of the expected parameters in the IBM i program (String type)
6. SEP: the separator used to delimit parameters and sizes (String type)
7. LOG: used only for working in progress (String type)

Running an existing program on IBM i with input and return parameters

PROCESS_RETURN_i function

This function executes a program developed on the IBM i whatever the language. Note that S36 is not supported.

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.PROCESS_RETURN_iAsync(Idt, BIB, PGM, PRM, VALPRM, SEP, RET,
LOG);
}
```

This function executes the program PGM of the BIB library with parameter passing and retrieves parameters after the execution.

The message sent by the IBM i at the end of the execution is placed in the variable “ErrorMsg” if failure.

Parameters (PRM) are sent to the program with their respective lengths (VALPRM) as expected in the program. Values and parameters are separated by the value contained in SEP.

The second last parameter of the list (here RET) indicates the number of parameters wanted in return. The values of these parameters in return are placed in the variable here called PRM if success.

The Error label and the requested parameters values are returned in the variable “ErrorMsg”.

8 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. BIB: the library that contains the program (String type)
3. PGM: the name of the program to launch (String type)
4. PRM: the parameter(s) in the form of a variable sent when launching the program, received back (type String).
5. VALPRM: the respective size of the expected parameters in the IBM i program (String type)
6. SEP: the separator used to delimit parameters and sizes (String type)
7. RET: receives parameters requested in return (String type)
8. LOG: used only for working in progress (String type)

Database records manipulation

Executing an SQL Query

ExecSQL function

This function is used to execute any SQL queries. The ErrMsg variable in return receives the result of the running of the query execution.

Syntax:

```
'All SQL queries
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.ExecSQLAsync(Idt, RqSQL, SessionId, "LKWORK");
}
```

4 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. RqSQL: query to execute (String type)
3. SessionId: contains the current HTML session ID (String type)
4. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

Executing more than one SQL Query in a single operation

ExecSQLbyTEN function

This function allows executing up to 10 SQL queries in a single operation.

Queries must be separated by the character '§'. The ErrMsg variable in return receives the result of the running of the query execution.

Syntax:

```
'Up to 10 SQL queries
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.ExecSQLbyTENAsync(Idt, RqSQL, SessionId, "LKWORK");
}
```

4 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. RqSQL: queries to execute separated by the character '§' (String type)
3. SessionId: contains the current HTML session ID (String type)
4. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

Number of records get by an SQL SELECT query

COUNT_OCCURS function

This function returns the number of occurrences of an SQL SELECT query.

The integer named ReturnVal retrieves the number of occurrences resulting from the SQL SELECT query (RqSQL) or the error message (success or failure).

Syntax:

```
{
public static int ReturnVal;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
ReturnVal = await Client1.COUNT_OCCURSAsync(Idt, RqSQL, "LKWORK");
}
```

The variable named NbrRcd retrieves the number of occurrences resulting from the SQL SELECT query (SelectQry). The variable ErrMsg (String type) retrieves the error message (success or failure).

3 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. RqSQL: the SQL SELECT query to count (String type)
3. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

Getting records of a table by an SQL SELECT query

GetDataAS400 function

The content of a table is recovered in an XML stream.

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.GetDataAS400Async(Idt, SelectQry, KeyFld, Base, Table, NbrRow,
Scenario, session, ErrMsg, BWork);
}
```

10 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. SelectQry: SQL SELECT query (simple or join) (String type)
3. KeyFld: primary key field(s) (String type)
4. Base: contains the name of the database of the file to display (String type)
5. Table: contains the name of the file to display (String type)

6. NbrRow: number of records of the table displayed (Integer type)
7. Scenario: Work Label (field) defined in the ASP.Net page, must have the following characteristics: `<asp:Label ID="LnkW" runat="server" style="z-index: 1; left: 1px; top: 1px; position: absolute; height: 0px; width: 0px;" Visible="False"></asp:Label>` (String type)
8. session: contains the current HTML session ID (String type)
9. ErrMsg: message in return for error or smooth execution (String type)
10. BWork: this parameter refers to the IBM i library that will store the temporary data needed to execute the function (String type)

Composition of a display for *UpdateRecord1 function*

The result is a serialized XML stream providing the detail of data and elements needed to display a record for an update operation (see [*UpdateRecord2XML*] function). The layout, the dimensioning, the constraints (Check and functional integrity), the identifiers, the respect of the natures of field are managed.

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.UpdateRecord1Async(Idt, KeyFld, RqSQL, SessionId, "LKWORK");
}
```

5 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. KeyFld: primary key field(s) (String type)
3. RqSQL: SQL SELECT query (simple or join) (String type)
4. SessionId: contains the current HTML session ID (String type)

5. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

CreateRecord1 function

The result is a serialized XML stream, containing a base object and providing the detail of elements needed to compose a display for a create operation (see [*CreateRecord2XML*] function). The layout, the dimensioning, the constraints (Check and functional integrity), the identifiers, the respect of the natures of field are managed. The function also returns the type of operation requested on the record.

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.CreateRecord1Async(Idt, KeyFld, RqSQL, SessionId, "LKWORK");
}
```

5 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. KeyFld: primary key field(s) (String type)
3. RqSQL: SQL SELECT query (simple or join) (String type)
4. SessionId: contains the current HTML session ID (String type)
5. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

DeleteRecord1 function

The result is a serialized XML stream, containing a base object and providing the detail of elements needed for displaying a record before a delete operation (see

[*DeleteRecord2XML*] function). The function also returns the type of operation requested on the record.

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.DeleteRecord1Async(Idt, KeyFld, RqSQL, SessionId, "LKWORK");
}
```

5 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. KeyFld: primary key field(s) (String type)
3. RqSQL: SQL SELECT query (simple or join) (String type)
4. SessionId: contains the current HTML session ID (String type)
5. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

DisplayRecord function

The result is a serialized XML stream, containing a base object and providing the detail of elements needed for a simple display of record (fields are locked).

Syntax:

```
{
public static string XMLString;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLString = await Client1.DisplayRecordAsync(Idt, KeyFld, RqSQL, SessionId, "LKWORK");
}
```

5 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. KeyFld: primary key field(s) (String type)
3. RqSQL: SQL SELECT query (simple or join) (String type)

4. SessionId: contains the current HTML session ID (String type)
5. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

Exploitation of the display of a database

UpdateRecord2XML function

Performs the update of the data displayed and modified by the function [*UpdateRecord1*].

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLUpdt = XMLUpdt.Replace("#", ((char)34).ToString());
ErrMsg = await Client1.UpdateRecord2XMLAsync(Idt, XMLUpdt, SessionId, "LKWORK");
}
```

4 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. XMLUpdt: describes how to update data
3. SessionId: contains the current HTML session ID (String type)
4. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

The string of characters “ErrMsg” provides the result of the function (failure or success).

CreateRecord2XML function

Creates a record entered and validated by the function [*CreateRecord1*].

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
```

```
XMLCrt = XMLCrt.Replace("#", ((char)34).ToString());
ErrMsg = await Client1.CreateRecord2XMLAsync(Idt, XMLCrt, SessionId, "LKWORK");
}
```

4 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. XMLCrt: describe how to create data.
3. SessionId: contains the current HTML session ID (String type)
4. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

The string of characters “ErrMsg” provides the result of the function (failure or success).

DeleteRecord2XML function

Deletes a record displayed and selected by the function [*DeleteRecord1*].

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
XMLDlt = XMLDlt.Replace("#", ((char)34).ToString());
ErrMsg = await Client1.DeleteRecord2XMLAsync(Idt, XMLDlt, SessionId, "LKWORK");
}
```

4 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. XMLDlt: describe how to delete data.
3. SessionId: contains the current HTML session ID (String type)
4. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

The string of characters “ErrMsg” provides the result of the function (failure or success).

Translating from EBCDIC to ASCII

ASCII_EBCDIC (all characters)

ASCII_EBCDIC function

This function translates an EBCDIC string into ASCII. It is used to solve the problem of residual special characters.

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
ErrMsg = await Client1.ASCII_EBCDICAsync(Ebcdic);
}
```

The string of characters ErrMsg retrieves the translated content of the Ebcdic string.

ASCII_EBCDIC (except “;”)

ASCII_EBCDICWithout function

This function translates an EBCDIC character string into ASCII except the ";" character. It is used to solve the problem of residual special characters in CSV files.

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
ErrMsg = await Client1.ASCII_EBCDICWithoutAsync(Ebcdic);
}
```

The string of characters ErrMsg retrieves the translated content of the Ebcdic string.

DATE CONVERSION

CVT_DATE function

This function translates a date of format "DATE" DB2 for i to the .Net date format.

Syntax:

```
Dim DateDB2 As String = "DATE IBM DB2 for i to convert"
Dim Return As String = ""
Return = x.CVT_DATE(DateDB2)
```

Return retrieves the translated content of the DateDB2 string.

TIME CONVERSION

CVT_TIME function

This function translates a date of format "TIME" DB2 for i to the .Net time format.

Syntax:

```
Dim TimeDB2 As String = "TIME IBM DB2 for i to convert"
Dim Return As String = ""
Return = x.CVT_TIME(TimeDB2)
```

Return retrieves the translated content of the TimeDB2 string.

TIMESTAMP CONVERSION

CVT_TIMESTAMP function

This function translates a date of format "TIMESTAMP" DB2 for i to the .Net time format.

Syntax:

```
Dim TimeStampDB2 As String = "TIMESTAMP IBM DB2 for i to convert"
```

```
Dim Return As String = ""
Return = x.CVT_TIMESTAMP(TimeStampDB2)
```

Return retrieves the translated content of the TimeStampDB2 string.

Using IBM i command in .Net environment

Retrieving IBM i command in XML format

PromptCMD function

This function retrieves a CL command in an XML stream, result of the CRCMDXML extraction program launch. In case of failure, the character string ErrMsg contains the error message.

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
ErrMsg = await Client1.PromptCMDAsync(Idt, CMD, SessionId, Library, "CRCMDXML");
}
```

In this case, the *PromptCMD* function retrieves the CMD CL command in the ErrMsg character string in XML format, result from the CRCMDXML extraction program launch. In case of failure, the ErrMsg variable contains the error message.

5 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. CMD: the name of the CL command to extract (String type)
3. SessionId: contains the current HTML session ID (String type)
4. Library: name of the library containing the command extraction program (String type)
5. This parameter contains the name of the CLP program which extracts the CL command (here "CRCMDXML") (String type)

All source codes provided below are to be installed on your IBM i, in the library of your choice. The name of the library will have to be replaced in the CLP program named

CRCMDXML. Likewise, any modification of program names will have to be reflected in the calling source code(s) concerned. These programs are written in RPG language, you can do the same in COBOL.

CRCMDXML: CLP program submitted by the *PromptCMD* function

```
             PGM        PARM(&COM &SESS)
             DCL        VAR(&COM) TYPE(*CHAR) LEN(10)
             DCL        VAR(&SESS) TYPE(*CHAR) LEN(8)
             DCL        VAR(&FIC1) TYPE(*CHAR) LEN(10)
             DCL        VAR(&FIC2) TYPE(*CHAR) LEN(10)
             DCL        VAR(&PR) TYPE(*CHAR) LEN(20)
             ADDLIBLE   LIB(NEWRMF)
             MONMSG     MSGID(CPF0000)
             CHGVAR     VAR(&PR) VALUE(&COM *cat 'QSYS      ')
             CHGVAR     VAR(&FIC1) VALUE('F' *CAT &SESS *cat '1')
             CHGVAR     VAR(&FIC2) VALUE('F' *CAT &SESS *cat '2')
             CPYF       FROMFILE(PFRCMDN1) TOFILE(NEWRMF/&FIC1) +
                          MBROPT(*ADD) CRTFILE(*YES)
             CPYF       FROMFILE(PFRCMDN2) TOFILE(NEWRMF/&FIC2) +
                          MBROPT(*ADD) CRTFILE(*YES)
             CLRPFM     FILE(&FIC1)
             CLRPFM     FILE(&FIC2)
             OVRDBF     FILE(PFRCMDN1) TOFILE(&FIC1)
             OVRDBF     FILE(PFRCMDN2) TOFILE(&FIC2)
             CALL       PGM(NEWRMF/QRCMDNET) PARM(&PR)
             CALL       PGM(NEWRMF/QRCMDNET1)
             CALL       PGM(NEWRMF/QRCMDNET2)
ENDPGM
```

PFRCMDN1: First physical file for working in progress

```
      A          R FORMAT1
      A            CH1         8150A
      A            CH2         8150A
      A            CH3         8150A
      A            CH4         8150A
```

PFRCMDN2: Second physical file for working in progress

```
      A          R FORMAT2
      A            CH5         8150A
      A            CH6         8150A
      A            CH7         8150A
      A            CH8         8150A
```

PFRCMDN1 and PFRCMDN2 are physical files used by all programs, storing temporary results. They receive the XML data to compose the CL command.

QRCMDNET: RPG program called by the CRCMDXML CLP program (extraction)

```
Fpfrcmdn1  o  a e             disk
Fpfrcmdn2  o  a e             disk
**-- API Error Data Structure: --------------------------**
DApiError         Ds
D AeBytPro                      10i 0 Inz(%Size(ApiError))
D AeBytAvl                      10i 0
D                                1a
D AeExcpId                       7a
D AeExcpDta                    126a
**-- Global variables; ----------------------------------**
D OutStrLenRt     s             10i 0
D NotSup          s             10i 0
D FB              s             10i 0 Dim(3)
**-- Command return variable: --------------------------**
D CdCmdd0100      Ds
D CdBytRtn                      10i 0
D CdBytAvl                      10i 0
D CdCmdXml                   65200a
**-- Retrieve Command Text: ----------------------------**
D RtvCmdTxt       Pr                  ExtPgm('QCDRCMDD')
D RcCmdNamQ                     20a   Const
D RcDst                         10i 0 Const
D RcDstFmt                       8a   Const
D RcRcvVar                   65200a   Options(*VarSize)
D RcRcvFmt                       8a   Const
D RcError                    65200a   Options(*VarSize)
**-- Convert String: ----------------------------------**
D CvtString       Pr                  ExtPgm('QTQCVRT')
D CsInpCcsId                    10i 0 Const
D CsInpStrTyp                   10i 0 Const
D CsInpStr                   65200a   Options(*VarSize)
D CsInpStrSiz                   10i 0 Const
D CsOutCcsId                    10i 0 Const
D CsOutStrTyp                   10i 0 Const
D CsOutCvtAlt                   10i 0 Const
D CsOutStrSiz                   10i 0 Const
D CsOutStr                   65200a   Options(*VarSize)
D CsOutStrLenRt                 10i 0
D CsNotSup                      10i 0
D CsFB                          10i 0 Dim(3)
**
**-- Mainline: ----------------------------------------**
**
C     *entry        plist
C                   parm                    cmdbib           20
C                   CallP     RtvCmdTxt(cmdbib
C                                :%Size( CdCmdd0100 )
C                                :'DEST0100'
```

```
C                                  :CdCmdd0100
C                                  :'CMDD0100'
C                                  :ApiError
C                                  )
C**
C*                    CallP     CvtString( 1208
C*                                 :0
C*                                 :CdCmdXml
C*                                 :CdBytRtn
C*                                 :37
C*                                 :0
C*                                 :0
C*                                 :CdBytRtn
C*                                 :CdCmdXml
C*                                 :OutStrLenRt
C*                                 :NotSup
C*                                 :FB
C*                                 )
C                     movel     *blanks       ch1
C                     movel     *blanks       ch2
C                     movel     *blanks       ch3
C                     movel     *blanks       ch4
C                     movel     *blanks       ch5
C                     movel     *blanks       ch6
C                     movel     *blanks       ch7
C                     movel     *blanks       ch8
**
 /free
   ch1 = %subst(CdCmdd0100 : 9 : 8150);
   ch2 = %subst(CdCmdd0100 : 8159 : 8150);
   ch3 = %subst(CdCmdd0100 : 16309 : 8150);
   ch4 = %subst(CdCmdd0100 : 24459 : 8150);
   ch5 = %subst(CdCmdd0100 : 32609 : 8150);
   ch6 = %subst(CdCmdd0100 : 40759 : 8150);
   ch7 = %subst(CdCmdd0100 : 48909 : 8150);
   ch8 = %subst(CdCmdd0100 : 57059 : 8150);
 /end-free
c                     movel     *blanks       CdCmdd0100
c                     movel     ch1           z20               9
c*    z20             ifeq      '<QcdCLCmd'
c                     write     format1
c                     write     format2
c*                    endif
c                     seton                                       lr
```

QRCMDNET1: RPG program called by the CRCMDXML CLP program (formatting part 1)

```
     Fpfrcmdn1  up   e             disk
```

```
D* compile with DFTACTGRP(*NO)
D* sample program assumes that the *SRCF is CCSID(37) as variable
D*        TestData values are imbedded in the program source
D* no error checking is provided (if you give "bad" CCSIDs to
D*        QTQCVRT)
D*
D* Mapping tables
D*
DStartMap         s            256
DTo819            s            256
D*
D* Data variables
D*
DTestData         s           8150
D*
D* Parameters for QTQCVRT/CDRCVRT API, see API Reference for details
D*
DCCSID1           s             10i 0 inz(1208)
DST1              s             10i 0 inz(0)
DL1               s             10i 0 inz(%size(StartMap))
DCCSID2           s             10i 0 inz(37)
DST2              s             10i 0 inz(0)
DGCCASN           s             10i 0 inz(0)
DL2               s             10i 0 inz(%size(To819))
DL3               s             10i 0
DL4               s             10i 0
DFB               s             12
D*
D* Prototype the MI instruction
D*
DConvert          PR                  EXTPROC('_XLATEB')
D                                  *   VALUE
D                                  *   VALUE
D                                10u 0 VALUE
D*
D* Working area to get hex values
D*
D                 ds
D x                               5i 0
D LowX                   2        2
C*
C* Initialize StartMap from x'00' to x'FF'
C*
C     0             do        255           x
C                   eval      %subst(StartMap:x+1:1) = LowX
C                   enddo
C*
C* Now get translate table values for CCSID 37 to CCSID 819
```

```
C*
C                   call      'QTQCVRT'
C                   parm                    CCSID1
C                   parm                    ST1
C                   parm                    StartMap
C                   parm                    L1
C                   parm                    CCSID2
C                   parm                    ST2
C                   parm                    GCCASN
C                   parm                    L2
C                   parm                    To819
C                   parm                    L3
C                   parm                    L4
C                   parm                    FB
C*
C* Now convert 'This is test data' to 819 from 37 using MI
C*
C                   callp     Convert( %addr(ch1)
C                                      :%addr(To819)
C                                      :%size(ch1))
C                   callp     Convert( %addr(ch2)
C                                      :%addr(To819)
C                                      :%size(ch2))
C                   callp     Convert( %addr(ch3)
C                                      :%addr(To819)
C                                      :%size(ch3))
C                   callp     Convert( %addr(ch4)
C                                      :%addr(To819)
C                                      :%size(ch4))
c                   update    format1
```

QRCMDNET2: RPG program called by the CRCMDXML CLP program (formatting part 2)

```
Fpfrcmdn2  up   e             disk
D* compile with DFTACTGRP(*NO)
D* sample program assumes that the *SRCF is CCSID(37) as variable
D*        TestData values are imbedded in the program source
D* no error checking is provided (if you give "bad" CCSIDs to
D*        QTQCVRT)
D*
D* Mapping tables
D*
DStartMap         s            256
DTo819            s            256
D*
D* Data variables
D*
DTestData         s           8150
```

```
D*
D* Parameters for QTQCVRT/CDRCVRT API, see API Reference for details
D*
DCCSID1           s             10i 0 inz(1208)
DST1              s             10i 0 inz(0)
DL1               s             10i 0 inz(%size(StartMap))
DCCSID2           s             10i 0 inz(37)
DST2              s             10i 0 inz(0)
DGCCASN           s             10i 0 inz(0)
DL2               s             10i 0 inz(%size(To819))
DL3               s             10i 0
DL4               s             10i 0
DFB               s             12
D*
D* Prototype the MI instruction
D*
DConvert          PR                  EXTPROC('_XLATEB')
D                                  *   VALUE
D                                  *   VALUE
D                                10u 0 VALUE
D*
D* Working area to get hex values
D*
D                 ds
D x                                5i 0
D LowX                    2        2
C*
C* Initialize StartMap from x'00' to x'FF'
C*
C     0             do        255           x
C                   eval      %subst(StartMap:x+1:1) = LowX
C                   enddo
C*
C* Now get translate table values for CCSID 37 to CCSID 819
C*
C                   call      'QTQCVRT'
C                   parm                    CCSID1
C                   parm                    ST1
C                   parm                    StartMap
C                   parm                    L1
C                   parm                    CCSID2
C                   parm                    ST2
C                   parm                    GCCASN
C                   parm                    L2
C                   parm                    To819
C                   parm                    L3
C                   parm                    L4
C                   parm                    FB
```

```
C*
C* Now convert 'This is test data' to 819 from 37 using MI
C*
C                   callp     Convert( %addr(ch5)
C                                      :%addr(To819)
C                                      :%size(ch5))
C                   callp     Convert( %addr(ch6)
C                                      :%addr(To819)
C                                      :%size(ch6))
C                   callp     Convert( %addr(ch7)
C                                      :%addr(To819)
C                                      :%size(ch7))
C                   callp     Convert( %addr(ch8)
C                                      :%addr(To819)
C                                      :%size(ch8))
c                   update    format2
```

Creating a prompt to use IBM i command

DoPromptCmdAS400 function

This function composes the prompt of a previously extracted CL command in XML format.

Syntax:

```
Dim i As Integer = 0
Dim XML_Prompt As String = " "
XML_Prompt = x.DoPromptCmdAS400(xml)
```

In this case, the *DoPromptCmdAS400* function composes an input prompt in an XML string (here XML_Prompt) of the CL command contained in the xml variable previously composed by the *PromptCMD* function.

1 parameter is required:

1. xml : contains the CL command to transform into an input prompt (String type)

Creating a formatted IBM i command from a prompt

DoCmdAS400 function

This function composes a CL command ready to be executed by the function *RUNCMD* from the prompt composed by the *DoPromptCmdAS400* function.

Syntax:

```
Cmd1 = x.DoCmdAS400(i, XML_Cmd)
```

In this case, the *DoCmdAS400* function composes an executable CL command from the XML string (here XML_Cmd) input prompt in the return string “Cmd1”. The variable "`i`" contains the number of topics generated in the prompt.

2 parameters are required:

1. `i`: numeric variable containing the number of topics generated in the prompt (Integer type)
2. XML_Cmd: XML string input prompt composed by the *DoPromptCmdAS400* function.

Export Data

Export in text files

FreeTXT function

This function exports the result of an SQL SELECT query into one or more tables in TXT format. In the case of a large query, when the Text export document reaches 1,048,576 rows, the function automatically closes it, allocates a subscript to it, and reopens a new one. The function returns the result of its execution in a string of characters (here “ErrMsg”).

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
ErrMsg = await Client1.FreeTXTAsync(Idt, RqSQL, Path, File, SessionId, "LKWORK");
}
```

6 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. RqSQL: SQL SELECT query to execute (String type)
3. Path: contains the backup path of Text file(s) for export (String type)
4. File: name of the table(s) result of the SQL SELECT query (String type)
5. SessionId: contains the current HTML session ID (String type)
6. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

Export in Excel format (XLSX)

FreeXLS function

This function exports the result of an SQL SELECT query into one or more Excel tables in XLSX format. In the case of a large query, when the Excel export document reaches 1,048,576 rows, the function automatically closes it, allocates a subscript to it, and reopens a new one. The function returns the result of its execution in a string of characters (here “ErrMsg”).

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
ErrMsg = await Client1.FreeXLSAsync(Idt, RqSQL, Path, File, SessionId, "LKWORK");
}
```

6 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. RqSQL: SQL SELECT query to execute (String type)
3. Path: contains the backup path of Excel file(s) for export (String type)
4. File: name of the table(s) result of the SQL query (String type)
5. SessionId: contains the current HTML session ID (String type)
6. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

Export in CSV format

FreeCSV function

This function exports the result of an SQL SELECT query into one or more tables in CSV format. In the case of a large query, when the CSV export document reaches 1,048,576 rows, the function automatically closes it, assigns it a hint, and reopens a new one. The function returns the result of its execution in a string of characters (here “ErrMsg”).

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
ErrMsg = await Client1.FreeCSVAsync(Idt, RqSQL, Path, File, SessionId, "LKWORK");
}
```

6 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. RqSQL: SQL SELECT query to execute (String type)
3. Path: contains the backup path of CSV file(s) for export (String type)
4. file: name of the table(s) result of the SQL SELECT query (String type)
5. session: contains the current HTML session ID (String type)
6. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

Using EXCEL to work with DB2 for i

Creating an EXCEL document for updating data

XLSXforUPDATE function

This function exports the result of an SQL SELECT query into one or more Excel tables in XLSX format. The resulting Excel table(s) can be used as an interface for updating one or several database records (see [*ApplyXLSX*] object described later). The layout, the dimensioning, the constraints (Check and functional integrity), the identifiers, the respect of the natures of field are managed. To be efficient, an Excel export document is restricted to 100 rows. In the case of a large query (more than 100 rows), the function automatically closes the document, allocates a subscript to it, and reopens a new one. The function returns the result of its execution specifying the path and the number of tables created in a string of characters (here “ErrMsg”).

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
ErrMsg = await Client1.XLSXforUPDATEAsync(Idt, RqSQL, KeyFld, Path, file, Base, Table,
SessionId, "LKWORK");
}
```

9 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. RqSQL: SQL SELECT query to execute (String type)
3. KeyFld: field(s) used to add an ORDER BY to the SQL query (String type)
4. Path: contains the backup path of Excel file(s) for export (String type)
5. file: contains the name of the IBM i original file (String type)
6. Base: contains the name of the database of the IBM i original file (String type)

7. Table: contains the name of the resulting Excel table(s) (String type)
8. SessionId: contains the current HTML session ID (String type)
9. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

Creating an EXCEL document for creating data

XLSXforCREATE function

This function exports the result of an SQL SELECT query into one Excel table in XLSX format, containing 100 blank rows with the name and type of the fields. The layout, the dimensioning, the constraints (Check and functional integrity), the identifiers, the respect of the natures of field are managed. In the case of more records creation wanted, the Excel export document can be duplicated by the operator. The resulting Excel table can be used as an interface for creating one or several database records (see [*ApplyXLSX*] object described later). The function returns the result of its execution specifying the path and the table created in a string of characters (here “ErrMsg”).

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
ErrMsg = await Client1.XLSXforCREATEAsync(Idt, RqSQL, KeyFld, Path, file, Base, Table,
SessionId, "LKWORK");
}
```

9 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. RqSQL: SQL SELECT query to execute (String type)
3. KeyFld: field(s) used to add an ORDER BY to the SQL query (String type)
4. Path: contains the backup path of Excel file for export (String type)
5. file: contains the name of the IBM i original file (String type)

6. Base: contains the name of the database of the original file (String type)
7. Table: contains the name of the resulting Excel table (String type)
8. SessionId: contains the current HTML session ID (String type)
9. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (String type)

XLSXforCREATEWithAGeneric function

The resulting Excel table can be used as an interface for creating one or several database records (see [*ApplyXLSX*] object described later). Its first row contains the requested generic record to be used as a pattern.The function retrieves the result of its execution specifying the path and the table created in a string of characters (here “ErrMsg”).

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
ErrMsg = await Client1.XLSXforCREATEWithAGenericAsync(Idt, RqSQL, KeyFld, Path, file,
Base, Table, SessionId, "LKWORK");
}
```

9 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. RqSQL: SQL SELECT query to execute (String type)
3. KeyFld: field(s) used to add an ORDER BY to the SQL query (String type)
4. Path: contains the backup path of Excel file for export (String type)
5. file: contains the name of the IBM i original file (String type)
6. Base: contains the name of the database of the original file (String type)
7. Table: contains the name of the resulting Excel table (String type)
8. SessionId: contains the current HTML session ID (String type)

9. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (String type)

Creating an EXCEL document for deleting data

XLSXforDELETE function

This function exports the result of an SQL SELECT query into one or more Excel tables in XLSX format. The resulting Excel table(s) can be used as an interface for deleting one or several database records (see [*ApplyXLSX*] object described later). To be efficient, an Excel export document is restricted to 100 rows. In the case of a large query (more than 100 rows), the function automatically closes the document, allocates a subscript to it, and reopens a new one. The function returns the result of its execution specifying the path and the number of tables created in a string of characters (here “ErrMsg”).

Syntax:

```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
ErrMsg = await Client1.XLSXforDELETEAsync(Idt, RqSQL, KeyFld, Path, file, Base, Table,
SessionId, "LKWORK");
}
```

9 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. RqSQL: SQL SELECT query to execute (String type)
3. KeyFld: field(s) used to add an ORDER BY to the SQL query (String type)
4. Path: contains the backup path of Excel file(s) for export (String type)
5. file: contains the name of the IBM i original file (String type)
6. Base: contains the name of the database of the original file (String type)
7. Table: contains the name of the resulting Excel table (String type)
8. SessionId: contains the current HTML session ID (String type)

9. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (here "LKWORK") (String type)

Applying an EXCEL document to DB2 for i

ApplyXLSX function

This function applies the action requested by the function at the origin of the resulting Excel table considered, on an IBM i database file. The function returns the result of its execution in a string of characters (here “ErrMsg”).

Syntax:

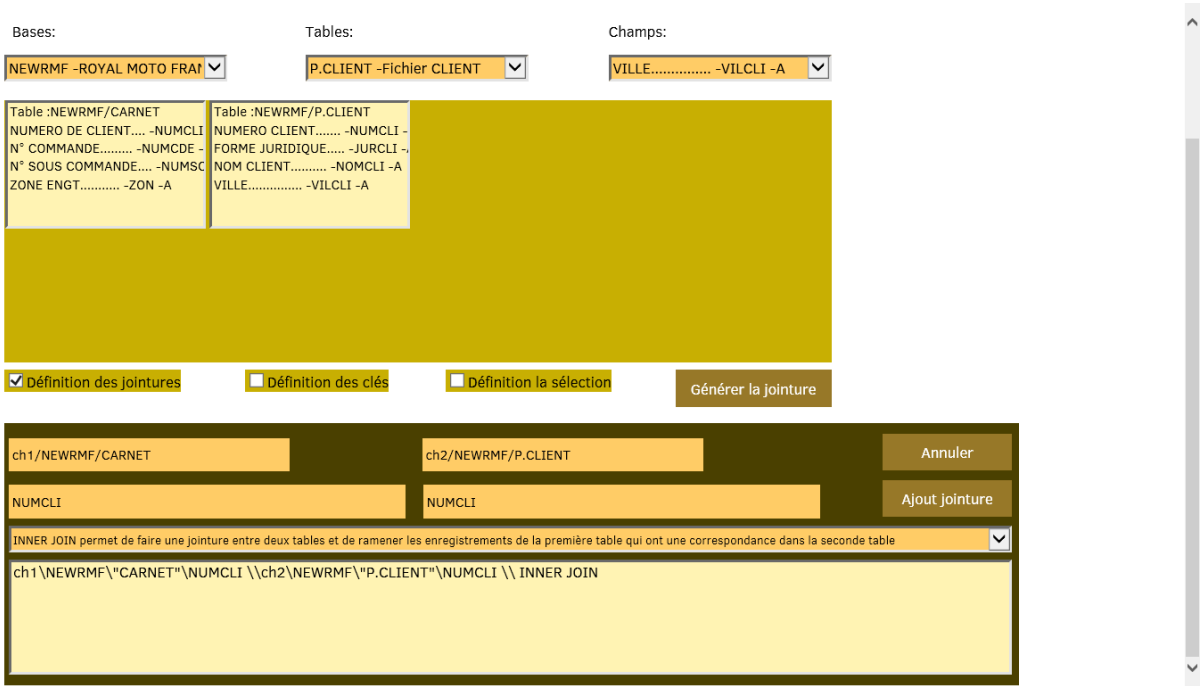
```
{
public static string ErrMsg;
ServiceReference1.Service1Client Client1 = new ServiceReference1.Service1Client();
ErrMsg = await Client1.ApplyXLSXAsync(Idt, Pathfile, Retour, "LKWORK");
}
```

4 parameters are required:

1. Idt: contains the encrypted identifier (String type)
2. Pathfile: contains the backup path and the Excel file name to be processed (String type)
3. Retour: contains the message sent by IBM i at the end of execution (success or failure) (String type)
4. This parameter refers to the IBM i library that will store the temporary data needed to execute the function (String type)

Appendix 1

A joins generator in SQL is supplied with Link2iFor.Net Core.



AUMERIAL SAS
2 rue Gustave EIFFEL
10430 ROSIERES
France
RCS Troyes 809 900 723
TVA FR9809900723