INTOGRATE NAV – Developer

Lotus Notes Integration Kit

For Dynamics NAV

Version 2.3.1

1. Introduction

1.1 INTOGRATE NAV to Lotus Notes integration

INTOGRATE NAV - Developer enables close integration between Dynamics NAV and Lotus Notes.

The technology applied in INTOGRATE NAV is based on integration to Lotus Notes via a Windows DLL program library and COM-technology. All administration and set-up tasks are carried out in NAV.

INTOGRATE features:

- **On-line integration functionality**
  All NAV updates can be coordinated with Lotus Notes updates. It is possible to cancel updates in case of e.g. a missing connection.
- **Strict observance of security routines in both systems**
  It is not possible to evade system security routines, neither in Lotus Notes, nor in NAV.
- **Embedding in standard NAV code**
  Coding is done directly in the NAV development environment using NAV methods. This ensures that NAV's rules of integrity are followed.
- **Transparent access to Lotus Notes**
  Lotus Notes can be accessed immediately with no need for coding or changes in Lotus Notes design-elements[1]. This enables e.g. integration with Lotus Notes applications for which the user has no access to source code but knows only the names of databases, forms and fields in Notes.
- **Sending of e-mails from NAV**
  It is possible to integrate with the E-mail system, which will enable e.g. work-flow systems to be implemented in NAV.
- **Direct access to Lotus Notes API**
  Because INTOGRATE has been developed specifically for Lotus Notes, it provides many Lotus Notes specific options and types of information.

INTOGRATE is also available for the applications Axapta and CONCORDE XAL.

1.2 Disclaimer

To prevent any side effects in your existing NAV any changes to your Dynamics NAV installation should be verified by the authority responsible for your NAV application (normally your NAV vendor). This includes hot fixes, patches, upgrades etc.

[1] Under certain circumstances, sending reports from NAV requires a minor modification of the Lotus Notes mail template.

1.3 Target audience

As the integration kit (INTOGRATE Developer) contains a collection of tools for integration with Lotus Notes via NAV's development environment, this manual is primarily intended for personnel with some knowledge of NAV's programming facilities.

However, the level of programming knowledge required much depends on the degree of integration wanted between NAV and Lotus Notes.

In order to use INTOGRATE Developer, some knowledge of Lotus Notes is of advantage, especially as regards the names of

- Databases
- Views
- Forms
- Fields

This information is required for in-house programming of integration solutions. However, skills in using the tools that supply this information from Lotus Notes can be acquired in a very short time. It is recommended that data transfer and manipulation in Lotus Notes be approved by the person(s) responsible for the company's Lotus Notes application(s).

1.4 Terminology

In the following sections, elements of NAV, references to menus, etc. are referred to as described in the below table:

<KEY>	Refers to a key on the keyboard. For example, <ENTER> means that you must press the Enter key.
'Menu\sub-menu\menu item'	Refers to a menu selection. For example, *'File\Open'* means that you must click *'Open'* in the *'File'* menu.
"c:\Lotus\Notes"	Refers to a path to either a file or a directory. Notice that the style resembles that of a menu selection.
NAVElement	NAV elements are referred to in italics, e.g. the code unit *Mail*.
[button]	Refers to a button on the screen. For example, [OK] refers to a button with the text 'OK'.
<u>MethodOrProperty</u>	All references to methods and properties in the INTOGRATE Developer are highlighted as shown.

1.5 Components

INTOGRATE NAV consists primarily of two main components: a COM-object “ERP2LN_COM.DLL” and a Windows program library called "ERP2LN.DLL".

1.5.1 Windows program library ERP2LN.DLL

This is the core of INTOGRATE and it handles the communication with Lotus Notes. This happens via a Windows program library, a so-called DLL file (Dynamic Link Library), which is one of the essential buildings blocks in Microsoft Windows.

ERP2LN.DLL is what is termed an "API" by programming specialists (Application Programming Interface), i.e. a program-based entry to a program.

In principle, ERP2LN.DLL functions as if the workstation was linked to Lotus Notes as an ordinary user.

1.5.2 COM-object ERP2LN_COM.DLL

Since NAV has a limited way of utilizing external functionality (outside NAV), a “wrapper” for “ERP2LN.DLL” has been made. The wrapper does not change the way “ERP2LN.DLL” works, it only adds another way to use it.

A NAV programmer has the option to use an automation object in the NAV code. The COM-object supports this type and can therefore be used by the programmer as any other automation object.

1.6 Training

A high level of training is not required in order to use INTOGRATE Developer. A course will typically comprise the following areas:

1. The principles applying to the integration between NAV and Lotus Notes
2. Integration kit commands
3. Relevant Lotus Notes tools
4. Integration case in which a NAV application element is integrated with a Lotus Notes database.

The duration of the course depends on the participants' basic knowledge of Lotus Notes and the NAV development environment, and on the level of detail required in the discussion of the integration case.

2. Program installation

INTOGRATE should normally be installed from a workstation where Lotus Notes and NAV clients are installed. All descriptions and tasks are based on the assumption that they are carried out from a local workstation.

2.1 Before you start

2.1.1 System requirements

The following software is necessary:

- Microsoft Windows Workstation/Server, NT, 2000, XP, Vista, Windows 7, Windows 8.
- Lotus Notes client version 5.02c or newer
- Microsoft NAV Attain DK 3.60, Microsoft Business Solutions-NAV DK 3.70, Microsoft Business Solutions-NAV DK 4.0, Microsoft Dynamics NAV 5.0, Microsoft Dynamics NAV 2009 or Microsoft Dynamics NAV 2013/2015/2016 (NAV 2016 in 32 bit client).

2.2 Check list

Before you begin the installation of INTOGRATE, it is recommended that you do the following. This may prevent potential problems not related to the integration kit itself.

- Start your local Lotus Notes client and logon with you password if required. If any errors occur at this stage, make sure to solve them before you use the integration kit.
- In Lotus Notes, open the databases that you plan to access from NAV. This ensures that you have the proper access rights.
- Start NAV and make sure that you have a valid user name and password for NAV. Do not forget to close NAV before you continue with the installation.
- If you need e-mail capabilities: Start your local Lotus Notes client. Use the shortcut key <CTRL>+<M> to create a new e-mail. If an error occurs at this point, this must be fixed before the installation of INTOGRATE can continue. A correctly set-up Lotus Notes e-mail client from which e-mails can be sent is a prerequisite.

2.3 Installation of INTOGRATE

Insert the CD called “INTOGRATE NAV – Developer”. If the installation program does not start automatically, double-click ”setup.exe” in the root of the CD (Click ’Run’ in the Windows Start menu and type ”D:\setup.exe” if the CD-ROM drive is the D-drive).

Follow the installation guidelines on the screen. For a detailed description of the installation, see the whitepaper “Technical Guidance - INTOGRATE NAV.pdf”.

2.4 License file

To use INTOGRATE a license file must be installed.

This file can be obtained either by purchasing a license for INTOGRATE or by downloading a demo version from our website (http://www.intograte.com).

The license file must be copied to the directory where the product was installed (the directory where "erp2ln.dll" is located).

2.5 Installation in NAV

When INTOGRATE has been installed (as described in the previous section) it can be used in NAV with no need for further installation. Simply declare an automation variable in your NAV code that points to the ERP2LN COM-object (see the next section for a detailed description).

2.6 Installation in Lotus Notes

Usually there is no need to install anything in Lotus Notes. However, if there are plans to send reports and/or links by email from NAV a minor change may be required in Lotus Notes (depends on Lotus Notes and NAV version). For instructions on installing this modification, see the document “Manual - INTOGRATE NAV - Reports by e-mail.pdf”.

All security is controlled by Lotus Notes. The integration kit will have the same privileges as the currently selected user-id in Lotus Notes.

It is not a requirement that the Lotus Notes client is running on the workstation for NAV to communicate with Lotus Notes. Starting Lotus Notes before using the integration kit is, however, recommended.

2.6.1 Avoid promptings for password

Since NAV uses the local Lotus Notes client for e.g. sending e-mails, the user will be prompted for a password for Lotus Notes when starting NAV regardless of whether the Lotus Notes client has been started or not. This can be changed so that the entering of a password is not necessary if the Lotus Notes client has already been started. Start your Lotus Notes client and select the menu ’File\Tools\User ID’. Mark ’Don't prompt for a password from other Notes-based programs…’ Close the dialog box by clicking [OK]. After this, the user will not be prompted for a Lotus Notes password when starting NAV. (However, this requires Lotus Notes to be running in the background and the user to be logged in).

2.6.2 Disable visible notification

Lotus Notes is locked when a message indicating that new mail has arrived is being displayed. The integration kit can thus not access Lotus Notes as long as the message is shown.

To circumvent this it is recommended that visible notification in Lotus Notes is disabled. This can be done from the menu ’File\Preferences\User preferences…’. Select the ’Mail and News’ icon on the left side of the dialog box. Deselect ’Visible Notification’ in the ’Receiving’ section.

2.6.3 Precautions with Lotus Notes changes

Generally, INTOGRATE uses the current setup of the Lotus Notes Client. It reads most setup information in the active location document (Office, Island, Home etc.).

This means that if the current location document is Island, INTOGRATE will not be able to connect to any Lotus Domino servers.

If changes are made in the location document or a new location is selected (Office, Island, Home etc.), make sure to restart both Notes and NAV. This is also the case when switching user id.

3. Getting started

Before an in-depth description of the facilities is given, a small case will be presented in this section illustrating how the kit functions and can be used.

The case should primarily be regarded as an example of the principles of integration. The case is fully operational, but, for example, error handling is not incorporated.

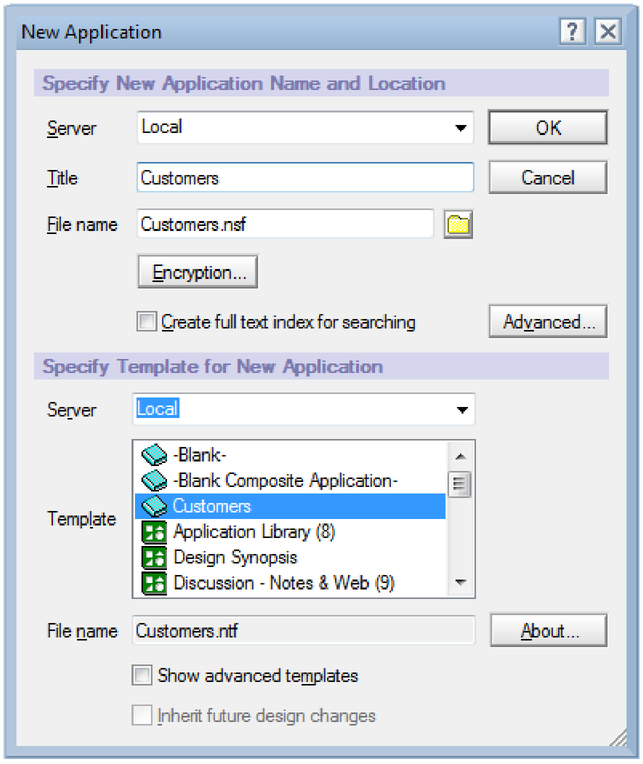
To use error-handling see section 4.6 Error handling0 on page 28.

3.1 Installing the Customer database

The installation case contains a template for creating a simple customer database in Lotus Notes. This can be found in the installation directory (typically "C:\Program Files\INTOGRATE NAV\Documentation\case").

The database is created as follows:

1. Copy the file "Customers.ntf" from the "case" directory to your local Lotus Notes client data directory (ex. "C:\Program Files\lotus\notes\data").
2. Start up your Lotus Notes client (if it is already running you might need to restart it).
3. Create a new database [CTRL] + [N].
4. Fill out the dialog as shown below.
5. It is important that "Customers CASE" is specified as Template.
6. Select OK.


New Application

Specify New Application Name and Location

Server: Local

Title: Customers

File name: Customers.nsf

☐ Create full text index for searching

Specify Template for New Application

Server: Local

Template: -Blank-, -Blank Composite Application-, Customers (selected), Application Library (8), Design Synopsis, Discussion - Notes & Web (9)

File name: Customers.ntf

☐ Show advanced templates

☐ Inherit future design changes

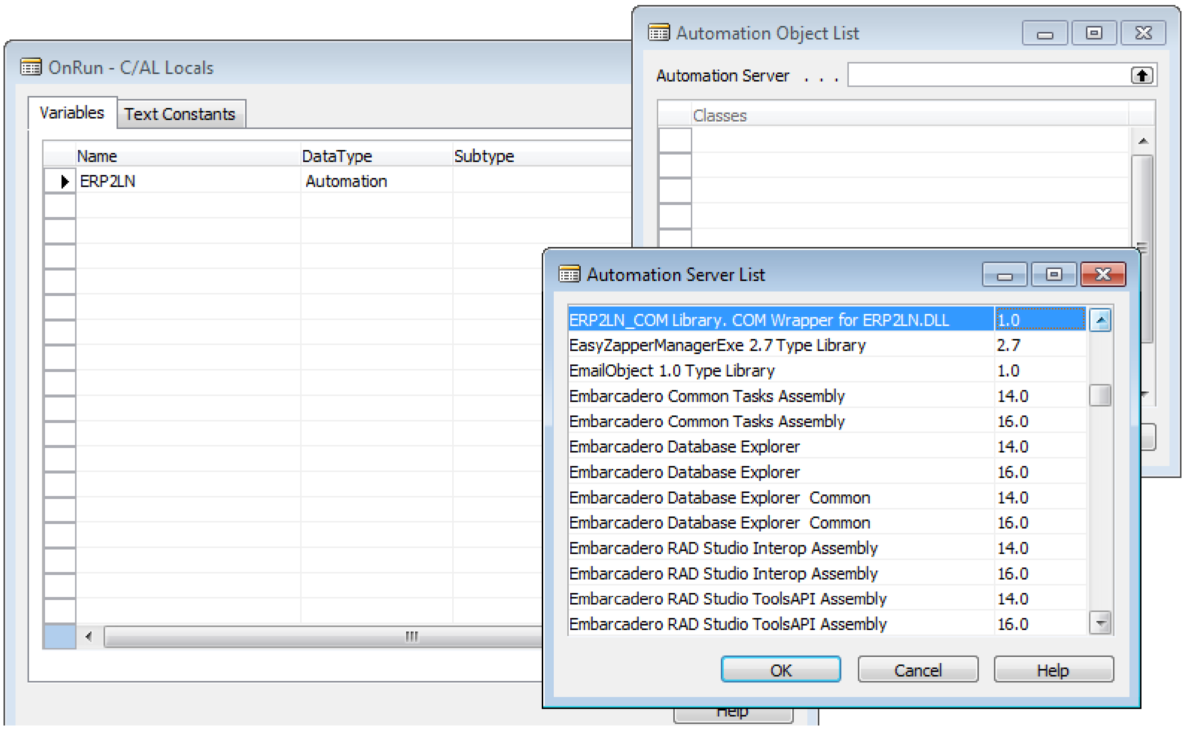
3.2 NAV code

Start by creating a new code unit and add the following local variables:

Name	Data type	Subtype	Length
ERP2LN	Automation	'ERP2LN_COM Library. COM Wrapper for ERP2LN.DLL'.ERP2LNCOM	
Customer	Record	Customer	
CommentLine	Record	Comment Line	
strDocId	Text		10

To add the automation object, use this procedure:

1. Browse the subtypes.
2. Browse the automation servers.
3. Select “ERP2LN_COM Library…” and click [OK].
4. A list of available objects is displayed. Select ERP2LNCOM library (see below)
5. Click OK to activate your choice.



Name	DataType	Subtype
ERP2LN	Automation	

ERP2LN_COM Library. COM Wrapper for ERP2LN.DLL	1.0
EasyZapperManagerExe 2.7 Type Library	2.7
EmailObject 1.0 Type Library	1.0
Embarcadero Common Tasks Assembly	14.0
Embarcadero Common Tasks Assembly	16.0
Embarcadero Database Explorer	14.0
Embarcadero Database Explorer	16.0
Embarcadero Database Explorer Common	14.0
Embarcadero Database Explorer Common	16.0
Embarcadero RAD Studio Interop Assembly	14.0
Embarcadero RAD Studio Interop Assembly	16.0
Embarcadero RAD Studio ToolsAPI Assembly	14.0
Embarcadero RAD Studio ToolsAPI Assembly	16.0

Use cut-and-paste to copy the following code into a new code unit in NAV.

```
CREATE(ERP2LN);
IF NOT ERP2LN.Init('Attain@' + SERIALNUMBER, 0) THEN
BEGIN
  ERROR('NotesInit error %1', ERP2LN.ErrorCode);
END;
ERP2LN.TruncateReturnString(1023);

ERP2LN.OpenServer('Local');
ERP2LN.Logon('Customers.nsf');
ERP2LN.Form := 'CustTable';
ERP2LN.View := 'CustTable';

IF NOT Customer.FIND('-') THEN EXIT; // No records found

REPEAT
  strDocId := ERP2LN.SearchView(Customer."No.");
  IF STRLEN(strDocId) = 0 THEN
  BEGIN // Customer not found
    ERP2LN.CreateNew(FALSE);
  END;

  ERP2LN.FieldValue('AccountNumber', Customer."No.");
  ERP2LN.FieldValue('Name', Customer.Name);
  ERP2LN.FieldValue('Address', Customer.Address);
  ERP2LN.FieldValue('City', Customer.City);
  ERP2LN.FieldValue('ZipCode', Customer."Post Code");
  ERP2LN.FieldValue('Country', Customer."Country/Region Code");
  ERP2LN.FieldValue('Attention', Customer."Name 2");
  ERP2LN.FieldValue('Phone', Customer."Phone No.");
  ERP2LN.FieldValue('Fax', Customer."Fax No.");

  Customer.CALCFIELDS(Balance);
  ERP2LN.FieldValue('Balance', FORMAT(Customer.Balance, 0, 2));

  CommentLine.SETFILTER("Table Name", 'Customer');
  CommentLine.SETFILTER("No.", Customer."No.");
  IF CommentLine.FIND('-') THEN
  BEGIN // Found one or more comment lines
    ERP2LN.FieldValue('Comments', FORMAT(CommentLine.Date) + ' ' + CommentLine.Comment);
    WHILE CommentLine.NEXT <> 0 DO
    BEGIN // For each subsequent line append it to the item
      ERP2LN.AppendTextList('Comments', FORMAT(CommentLine.Date) + ' ' +
        CommentLine.Comment);
    END;
  END;

  ERP2LN.Commit();
UNTIL Customer.NEXT = 0;
```

Save and execute the above code unit. This will create all the customers in NAV in the Lotus Notes database “Customers.nsf”. If the code unit is executed more than once, it will simply update existing customers in Lotus Notes.

3.3 Example walkthrough

First, the object used for communication with Lotus Notes is prepared:

```
CREATE(ERP2LN);
IF NOT ERP2LN.Init('Attain@' + SERIALNUMBER, 0) THEN
BEGIN
  ERROR('NotesInit error %1', ERP2LN.ErrorCode);
END;
```

Handle the limitation of a maximum of 1024 characters in any returned result in NAV:

```
ERP2LN.TruncateReturnString(1023);
```

Define the server to log on to (in this case log on to the local workstation):

```
ERP2LN.OpenServer('Local');
```

Log on to a database on the local workstation:

```
ERP2LN.Logon('Customers.nsf');
```

Specify the design-elements to use (usually required):

```
ERP2LN.Form := 'CustTable';
ERP2LN.View := 'CustTable';
```

Locate the first Customer in NAV (exit if no records found)

```
IF NOT Customer.FIND('-') THEN EXIT; // No records found
```

The repeat statement simply iterates all the customers in NAV.

In the repeat statement the first thing done is to search for the customer in Lotus Notes. If the customer record is not found it will be created by a call to the *CreateNew*:

```
strDocId := ERP2LN.SearchView(Customer."No.");
IF STRLEN(strDocId) = 0 THEN
BEGIN // Customer not found
  ERP2LN.CreateNew(FALSE);
END;
```

If the document was found by *SearchView* it will automatically be active in the ERP2LN object. This means that the next methods and properties called will work on this document.

The next block of code sets information on either the new document created by *CreateNew* or an existing found by *SearchView*.

```
ERP2LN.FieldValue('AccountNumber', Customer."No.");
ERP2LN.FieldValue('Name', Customer.Name);
ERP2LN.FieldValue('Address', Customer.Address);
ERP2LN.FieldValue('City', Customer.City);
ERP2LN.FieldValue('ZipCode', Customer."Post Code");
ERP2LN.FieldValue('Country', Customer."Country/Region Code");
ERP2LN.FieldValue('Attention', Customer."Name 2");
ERP2LN.FieldValue('Phone', Customer."Phone No.");
ERP2LN.FieldValue('Fax', Customer."Fax No.");
```

When transferring other values than text, they must first be converted. These lines first calculate the flow field “balance” and then convert it to text before updating the “Balance”-field on the document in Lotus Notes.

```
Customer.CALCFIELDS(Balance);
ERP2LN.FieldValue('Balance', FORMAT(Customer.Balance, 0, 2));
```

To add large text content to Lotus Notes an option is to create multi value items (several items with the same name in Lotus Notes) the method *AppendTextList* is used in combination with *FieldValue* to accomplish this:

```
CommentLine.SETFILTER("Table Name", 'Customer');
CommentLine.SETFILTER("No.", Customer."No.");
IF CommentLine.FIND('-') THEN
BEGIN // Found one or more comment lines
  ERP2LN.FieldValue('Comments', FORMAT(CommentLine.Date) + ' ' + CommentLine.Comment);
  WHILE CommentLine.NEXT <> 0 DO
  BEGIN // For each subsequent line append it to the item
    ERP2LN.AppendTextList('Comments', FORMAT(CommentLine.Date) + ' ' + CommentLine.Comment);
  END;
END;
```

4. Using INTOGRATE Developer

This section describes in more detail how to use the integration kit from NAV. For a complete reference of all methods and properties in the integration kit please see appendix 1.

All examples assume that the integration kit has been declared as the automation object 'ERP2LN' (see section 3.2 for an example).

4.1 Calling methods and properties

The integration kit works as any other automation object in NAV. This means that after it has been declared as a variable, properties and methods can be called the usual way.

The most typical way of using the automation object is like this (assuming that you declared the automation object as "ERP2LN"):

```
CREATE(ERP2LN);
IF NOT ERP2LN.Init('Attain@' + SERIALNUMBER, 0) THEN
BEGIN
  ERROR('NotesInit error %1', ERP2LN.ErrorCode);
END;
```

A useful function in NAV is the C/AL Symbol Menu where you can browse the accessible methods and properties of any automation object. This function can be activated by clicking [F5] or using the menu "View\C/AL Symbol Menu" while in the source code.

4.2 Connecting to Lotus Notes

Some of the integration kit commands concern the creation and closing of Lotus Notes sessions. Other commands are for the selection of application elements in Notes.

When *Init* is called from NAV, it must have the following syntax (this only changes if it is called from others systems such as Axapta, XAL etc.):

```
ERP2LN.Init('Attain@' + SERIALNUMBER, 0);
```

The SERIALNUMBER is a global method in NAV which returns the licensed serial number.

The *Init* method creates a link between NAV and the DLL library handling the system calls to Lotus Notes. This method must be called before any other integration methods can be used. Calling the method more than once will not have any effect (subsequent calls will be ignored).

4.2.1 Setting server and database

Before any connection is made a server name must be set. This is done by calling *OpenServer*.

To connect to the local machine, use:

```
ERP2LN.OpenServer('Local');
```

Connecting to a server is done in the same way. This example connects to the server Server1/Acme:

```
ERP2LN.OpenServer('Server1/Acme');
```

Having set the server name, you can log on to a database. This is done by calling *Logon*:

```
ERP2LN.Logon('Customers.nsf');
```

If the current user does not have access to this database or the database does not exist an error occurs. The system might also ask for the user's Lotus Notes password depending on the configuration (the password prompt is only requested once per NAV session). To disable the password prompt, see section 2.6.1.

The name of the database must be relative to the Notes Data directory.

4.2.2 Setting form and view

A Lotus Notes form describes the fields associated with a document and it must always be set.

An example is the 'Memo' form that describes all the fields on an e-mail in Lotus Notes:

```
ERP2LN.Form := 'Memo';
```

When the form is set a check is performed to verify if it exists in the currently selected database (set by calling *Logon*).

If you are using the lookup and query methods, a view must be selected. Call the property *View* to do this. A few examples:

```
ERP2LN.View := 'CustTable';
ERP2LN.View := '($All)';
```

4.2.3 Switching sessions

It is possible to operate with multiple sessions. Each session has its own connection to Lotus Notes, with server, database, form, view, current document etc.

To change between sessions, use the property *ActiveHandle*.

```
// Start by using default handle 1
ERP2LN.OpenServer('Server1/Acme');
ERP2LN.Logon('Customers.nsf');
ERP2LN.Form := 'CustTable';

intOriginalHandle := EPR2LN.ActiveHandle; // Save current handle

EPR2LN.ActiveHandle := 1; // Operate on a new handle
..
.. Work on handle 1
..

EPR2LN.ActiveHandle := 2; // Operate on a new handle
ERP2LN.OpenServer('Server2/Acme');
ERP2LN.Logon('Creditors.nsf');
ERP2LN.Form := 'CustTable';
..
.. Work on handle 2
..

EPR2LN.ActiveHandle := 1; // Switch back to handle 1
..
.. Work on handle 1
..

EPR2LN.ActiveHandle := intOriginalHandle; // Return to original handle
```

Be cautious when using multiple sessions. Confusion about which session is active is often responsible for errors.

4.3 Queries and lookup

4.3.1 Selecting a single document

For selecting a single document in Lotus Notes based on a unique key, the properties *UNID* and *DocId* can by used.

Often there is a need to store a unique key for a Lotus Notes Document in NAV. The only real unique key for a document in Lotus Notes is the Universal ID (also known as UNID). The UNID is a 32-character long hexadecimal string that uniquely identifies a document (across servers and databases).

The DocId is only unique in one database and should never be used for storing references to documents (It is not even unique across replicas of the same database). You only need DocId when using methods like *SearchView*, *SearchNext* and *Query* which all return DocIds.

A scenario for using the UNID could be that a customer record in NAV should have a link to its equivalent document in Lotus Notes. An example:

```
ERP2LN.OpenServer('Local');
ERP2LN.Logon('Customers.nsf');
ERP2LN.Form := 'CustTable';

IF NOT Customer.GET('60000') THEN ERROR('Customer not found!');
ERP2LN.UNID := Customer."NotesUNID"; // Lookup the document in Lotus Notes
IF ERP2LN.Error THEN ERROR('Document not found!');

MESSAGE(ERP2LN.FieldValue('Name'));
```

4.3.2 Queries

The method *Query* is used for execution of standard Notes queries. The selection criterion is specified as a parameter. The Parameter can be compared to "View selection" in Lotus Notes. In order to use this method, some knowledge of Lotus Notes is required.

Here are some examples of calling the method:

```
// returns all documents in the database.
intDocsFound := ERP2LN.Query('SELECT @ALL');

// Return all documents with a quantity higher than 100.
intDocsFound := ERP2LN.Query('SELECT Quantity>100');
```

By using the *Query* method, very flexible searches in Lotus Notes can be performed. But this method uses full-text queries in Lotus Notes and in large databases this method can be quite slow. An alternative lookup method is available in the method *SearchView* (see section 4.3.3 Iterating documents in a view).

To go through the result use the *MoveFirst*, *MoveNext*, *MovePrev* and *MoveLast* methods. The following example searches for all customers from Germany (DE). It then iterates and displays each customers name:

```
ERP2LN.OpenServer('Local');
ERP2LN.Logon('Customers.nsf');
ERP2LN.Form := 'CustTable';

intDocsFound := ERP2LN.Query('SELECT Country="DE"');
MESSAGE('Found ' + FORMAT(intDocsFound) + ' documents.');

IF intDocsFound  > 0 THEN
BEGIN
  ERP2LN.MoveFirst(); // Prepare first document
  REPEAT
    MESSAGE(ERP2LN.FieldValue('Name'));
  UNTIL STRLEN(ERP2LN.MoveNext()) = 0
END;
```

4.3.3 Iterating documents in a view by key(s)

The Query method uses full text searches which in many cases are relatively slow. To speed up iterating documents in Lotus Notes, views are typically used. A view with an index is comparable to a standard relational database in the sense that it is possible to use keys as criteria for which documents to iterate.

There are a few requirements of a view in Lotus Notes. First of all there must be at least one sorted column (a sorted column represents a key/index) and the sorted columns must be of type text. If a sorted column is not text, this can usually be fixed by creating a new view where the non-text value is converted using the “@Text()” formula.

To use more than one key (if there is more than one sorted column), simply separate the keys with ';' (the delimiter can be changed through the property *KeyDelimiter*).

The method *SearchView* is used for searching views. The method returns the first document that matches the search key(s). To iterate results from *SearchView,* use the method *SearchNext*.

The following example will iterate all documents in Lotus Notes where country is "DK" and zip code is "DK-8000":

```
ERP2LN.OpenServer('Local');
ERP2LN.Logon('Customers.nsf');
ERP2LN.Form := 'CustTable';

ERP2LN.View := 'CustomersByCountryAndZipCode';

strDocId := ERP2LN.SearchView('DK;DK-8000');
WHILE(StrLen(strDocId)) > 0 DO
BEGIN
  ERP2LN.GetFieldValue('Name', strName);
  MESSAGE(strName);

  strDocId := ERP2LN.SearchNext();
END;
```

If only documents where Country matches "DK" are needed, simply call *SearchView* like this:

```
strDocId := ERP2LN.SearchView('DK');
```

The number of sorted columns does not matter. The only requirement is that the supplied search key never contains more values than there are sorted columns in the Lotus Notes view.

4.3.4 Iterating all documents in a view

To iterate all documents in a view the method *SearchFirst* combined with *SearchNext* are used. The procedure is very similar to that of iterating a view by keys.

The following example will iterate all documents in the Lotus Notes view "CustTable":

```
ERP2LN.OpenServer('Local');
ERP2LN.Logon('Customers.nsf');
ERP2LN.Form := 'CustTable';

ERP2LN.View := 'CustTable';

strDocId := ERP2LN.SearchFirst();
WHILE(STRLEN(strDocId)) > 0 DO
BEGIN
  ERP2LN.GetFieldValue('Name', strName);
  MESSAGE(strName);

  strDocId := ERP2LN.SearchNext();
END;
```

4.4 Data manipulation

After establishment of a Lotus Notes session and maybe selection of a document for processing, data can be processed by means of different methods. These methods make it possible to create, edit or delete documents.

4.4.1 Creating and saving documents

The method *CreateNew* creates a new empty document in Lotus Notes. After a call to *CreateNew* the field values can be read and written using the methods described in section 4.4.2 Reading and setting values.

If the parameter for *CreateNew* is set to *true*, the new document will be saved immediately; otherwise it will not be saved until *Commit* is called.

Whenever a document has been modified it must be saved in Lotus Notes. This is done by calling *Commit* or *CommitWithForm*.

CommitWithForm will compute any "Computed Values" and execute formulas defined on the document's associated form. This is very similar to saving a document in the Lotus Notes client. It is considerably slower than *Commit*.

Commit saves all changes to the document. It does not execute any formulas or calculate any computed values.

If NEITHER method (*Commit* and *CommitWithForm*) is called, no changes are saved.

An example of using *Commit* can be found in section 3.2.

4.4.2 Reading and setting values

To get and set document values in Notes, use the method *GetFieldValue* to read data and the property *FieldValue* to write data. It is also possible to use the property *FieldValue* to read data, but NAV has some length issues that make it unsuitable for this purpose (they will be discussed later in this section).

All data sent and received through INTOGRATE must first be converted to text. When INTOGRATE receives the text it will automatically convert it to the appropriate data type in Lotus Notes (based on the active form).

A simple example of reading and writing values from the active document:

```
// Write a value in the Lotus Notes item "Name"
// using the value "Name" in the Customer table.
ERP2LN.FieldValue('Name', Customer."Name");

// Read a value into local variable "strAccountNumber"
// from the Lotus Notes item 'AccountNumber'.
ERP2LN.GetFieldValue('AccountNumber', strAccountNumber);
MESSAGE(strAccountNumber);
```

Since Lotus Notes is not a relational database a document can have items that are not described on the associated form. To read and write these items the method *GetItemValue* and property *ItemValue* must be used.

When using these methods INTOGRATE makes no attempt to verify if the given item exists on the selected form (no "data dictionary" check is performed). If possible, always use the method *GetFieldValue* and property *FieldValue* to get and set values.

When setting an item with the property *ItemValue*, the data type will be decided based on the following scenarios:

1. If the item name has a post-fixed data type this will be used (see later in this section for a description).
2. If the item exist on the document's form (it is defined in the design of the form in Lotus Notes) the specified data type of that field will used as data type. If, for example, an item named "Balance" exists on the form with type number, the stored value in Lotus Notes will be of type number.
3. If none of the above situations apply the data type will be text.

It is possible to force a specific data type regardless of what is defined on a form in Lotus Notes. Any item name can have a data type post-fixed like this:

```
ERP2LN.ItemValue('#NUMBER#Balance', Customer."Balance");
```

Details on the data types can be found in the development reference list in the section about the property *ItemValue*.

When the property *ItemValue* or *FieldValue* is used for reading data, no returned string may ever exceed 254 characters (again, a limitation in NAV). If the size is exceeded NAV will generate an error (that cannot be caught). To handle returned values larger than 254 characters use *GetItemValue* and *GetFieldValue* instead.

4.4.2.1 Large text content

To transfer large amounts of text (more than 1024 characters) from NAV to Lotus Notes the method *AppendItemValue* can be used.

The following example sets the initial value using the property *FieldValue* and then calls *AppendItemValue* for each block of text.

```
ERP2LN.FieldValue('Name', 'FirstBlock');
ERP2LN.AppendItemValue('Name', 'SecondBlock');
ERP2LN.AppendItemValue('Name', 'ThirdBlock');
```

The resulting value in the "Name" item will be "FirstBlockSecondBlockThirdBlock".

Beware that if the item being set is configured as a multi-value item, any presence of a delimiter in the value will be treated as a multi-value separator. The default delimiter is a semicolon ";" and it can be changed using the property *ListDelim*.

4.4.2.2 Text lists and multi-value items

To create text lists in Lotus Notes use the methods *AppendTextList*. Text lists are also referred to as multi-value text in Lotus Notes.

The following example creates a text list with 3 members:

```
ERP2LN.FieldValue('Name', 'FirstTextElement');
ERP2LN.AppendTextList('Name', 'SecondTextElement');
ERP2LN.AppendTextList('Name', 'ThirdTextElement');
```

To set data types other than text use *AppendItemValue* and separate the multi-values with a semi colon (or the delimiter configured using the *ListDelim* property).

This example assumes that a field named "DateList" is configured on the active form. The field is set as multi-value and must contain date/time values.

```
ERP2LN.FieldValue('DateList', FORMAT(TODAY, 0, '<Day,2>-<Month,2>-<Year4>'));
ERP2LN.AppendItemValue('DateList', ';' + '31-12-1999');
ERP2LN.AppendItemValue('DateList', ';' + '31-12-2000');
```

The result will be three multi-values of type date in the item named "DateList". It is possible to call *AppendItemValue* with several values:

```
ERP2LN.AppendItemValue('DateList', ';31-12-1999;31-12-2000;31-12-2001');
```

NAV does not understand multi-values in the same way as Lotus Notes does. When requesting such values from Lotus Notes, the values are returned delimited by semicolon (or another delimiter set by the property *ListDelim*).

4.4.2.3 Date/time format

All date and time values must have a special format before they are transferred to the integration kit.

Depending on whether both date and time are needed use the following table to decide on the format:

	Format	NAV format expression
Date only	dd-mm-yyyy	<Day,2>-<Month,2>-<Year4>
Time only	hh:mm:ss	<Hours24,2><Filler Character,0>:<Minutes,2>:<Seconds,2>
Date and time	dd-mm-yyyy hh:mm:ss	Above to expressions separated with a whitespace.

4.4.2.4 Number format

Any type of numbers transferred to Lotus Notes must be converted to text using the method *FORMAT* like this:

```
Customer.CALCFIELDS(Balance);
ERP2LN.FieldValue('Balance', FORMAT(Customer.Balance, 0, 1));
```

The above example assumes that the format type 1 is correctly configured. Format 1 should be specified as this NAV expression:

```
<Sign><Integer><Decimals><1000Character,,>
```

4.4.3 Deleting documents

The integration kit provides two methods for deleting documents in Lotus Notes. These are *DeleteDocumentByID* and *DeleteCurrentDocument*.

DeleteCurrentDocument simply deletes the active document and *DeleteDocumentByID* deletes a document by its id (DocId).

After deletion of a document via *DeleteCurrentDocument*, a new current document must be selected before further data manipulation can be done.

4.4.4 File attachments

The integration kit has various methods for handling attachments in Lotus Notes. These methods include attaching, detaching and deleting file attachments.

Files are attached to a Notes document via the method *AttachFile*. This method takes to parameters, a filename and optionally an attachment name. The first parameter specifies the file to attach (usually the full path to the file) and the second parameter, if set, decides the attachment name on the document.

If no other attachment name is specified, the attachment will be created with the same name as the file (excluding preceding path).

DetachFile works in the same way except that the filename is the resulting location of the attachment and the attachment name is the attachment to detach.

This simple example creates an attachment and immediately detaches it:

```
ERP2LN.AttachFile('c:\temp\report.doc', 'Annual report.doc');
ERP2LN.DetachFile('c:\temp\Annual report 2006.doc', 'Annual report.doc');
```

Beware that calling *DetachFile* will always overwrite any existing file.

To delete an attachment use the method *DeleteAttachment*. An example:

```
ERP2LN.DeleteAttachment('Annual report.doc');
```

This will delete the previously attached file “Annual report.doc”.

To delete all attachment to the current document call *DeleteAttachAll*.

When deleting attachments placed in rich-text items the icon representing the attachment is not removed.

4.5 Mail

There are several methods and properties in the integration kit that enable the sending of e-mails via the Lotus Notes client with and without user interaction.

Contrary to normal access to databases in Lotus Notes, mail actions do no require the user to call any of the methods *Logon*, *OpenServer*, *CreateNew* etc. This is done automatically based on the current user’s mail setup in Lotus Notes.

When creating and sending e-mails the following steps are usually taken:

1. Define the receivers and the subject of the e-mail (this optionally includes any cc and/or bcc receivers).
2. Build the body content of the mail.
3. Attach one or more files.
4. Send the mail in the background or open it in the Lotus Notes client.

4.5.1 Receivers and subject

To add receivers to the “To” field call *MailSendTo*. To set the “CC” and “BCC” fields use *MailCC* and *MailBCC*. These methods take one parameter containing one or more mail addresses (separated by a comma). Calling any of these methods more than once will result in a append operation (existing mail addresses will not be overwritten).

Use the property *MailSubject* to set the subject of the mail.

This example sets the “To”, “CC” and Subject of the next mail to send.

```
ERP2LN.MailSendTo('john@acme.com,jane@acme.com');
ERP2LN.MailCC('joe@acme.com');
ERP2LN.MailCC('carol@acme.com'); // Append’s to joe@acme.com

ERP2LN.MailSubject := 'January Newsletter 2006';
```

4.5.2 Body content

Setting the body content can be done in two ways either by using the simple method *MailBodyText* or the more complex methods *MailBody* and *MailBodyFile*. The first method can be used for text mails only (ASCII text). The second methods are used when the content needs to be formatted and possibly contain pictures (Lotus Notes rich text).

To create simple body content comprising only text, call *MailBodyText* one or more times. The following example builds up a body:

```
strName := 'John';
ERP2LN.MailBodyText('Dear ' + strName);
ERP2LN.MailBodyText(''); // Empty line
ERP2LN.MailBodyText('We are pleased....');
ERP2LN.MailBodyText(''); // Empty line
ERP2LN.MailBodyText('Kind regards...');
```

To build a more complex mail body containing for example both text and embedded pictures use *MailBody* and *MailBodyFile*.

The following example will accomplish the same as the above example but there will be a logo appended to the mail body:

```
ERP2LN.MailBody('Dear ' + strName);
ERP2LN.MailBody(''); // Empty line
ERP2LN.MailBody('We are pleased....');
ERP2LN.MailBody(''); // Empty line
ERP2LN.MailBody('Kind regards...');
ERP2LN.MailImportBodyFile('c:\temp\logo.jpg');
```

The method *MailImportBodyFile* can be used for several types of import. Most formats supported by Lotus Notes are also supported by *MailImportBodyFile*. To make sure the import works as expected, try sending a test mail to yourself before sending to other recipients.

The formats supported are always dictated by the installed Lotus Notes client.

4.5.3 Attach files

Attaching files to a mail works as attaching files to an ordinary document in Lotus Notes. To attach a file to the current mail call *MailAttachFile*.

This example attaches “c:\temp\report.doc” to the mail:

```
ERP2LN.MailAttachFile('c:\temp\report.doc', 'Annual report.doc');
```

MailAttachFile can be called several times to attach more than one file.

For more details see the 4.4.4 File attachments section found on page 24.

4.5.4 Sending a mail

When the mail has been prepared (setting recipients, subject, body and adding attachments) it can be sent either in the background or with user interaction.

To send backend (no user interaction) simply call *MailSend*. This is demonstrated in this example:

```
ERP2LN.MailSendTo('john@acme.com');
ERP2LN.MailSubject := ‘January Newsletter’;
ERP2LN.MailBodyText('Dear John');
ERP2LN.MailBodyText(''); // Empty line
ERP2LN.MailBodyText('We are pleased....');
ERP2LN.MailBodyText(''); // Empty line
ERP2LN.MailBodyText('Kind regards...');
ERP2LN.MailSend(‘’); // Send the mail in the background
```

To allow the user to edit the mail before sending it call *MailCompose* instead of *MailSend*. This will open the mail in Lotus Notes for editing. If recipients, subject etc. are set, these will automatically be filled in. *MailCompose* does not wait for the user to send the mail, it returns immediately. To get a reference to the mail call <u>*MailCompose*</u> with the parameter set to *true* (if the mail is not sent it will remain as a Draft mail in Lotus Notes).

4.5.4.1 Mail signature

In Lotus Notes a user has the option to append a signature to each mail. If this has been enabled it will also be honoured when sending mails from the integration kit.

To disable this feature set the property <u>*MailAppendSignature*</u> to false.

4.5.5 Automatic Error Messages by e-mail

The integration kit has a facility by which messages about errors occurring in connection with integration are mailed to e.g. the IT-responsible. This may facilitate internal support as the mail contains most of the relevant information. This facility is especially well suited when using the integration kit on a batch server where error messages may not necessarily be seen immediately.

The facility is activated by the method <u>*MailOnError*</u> in which the receiver is specified as a parameter. More receivers can be specified, separated by commas.

To enable this facility use the following code:

```
ERP2LN.MailOnError := 'john@acme.com';
```

To disable the feature simply call it like this:

```
ERP2LN.MailOnError := '';
```

Beware that if the mail setup in Lotus Notes is incorrect, this function will not work.

4.6 Error handling

Using error handling is usually recommended. Consider the following code example:

```
CREATE(ERP2LN);
ERP2LN.Init('Attain@' + SERIALNUMBER, 0);
..
Some other NAV code
..
```

If *Init* fails in the above example the NAV code following is still executed.

To avoid this, use the return value of *Init* to see if it was successful. In many cases a method returns either *true* or *false* indicating success or failure. It is also possible to read the property *Error* after a call to a method or property. If *Error* is *true* use the property *ErrorText* and *ErrorCode* to get more details of the occurred error.

See the developer reference in appendix 1 for details on the different error codes.

An example that shows different kinds of error handling:

```
CREATE(ERP2LN);
IF NOT ERP2LN.Init('Attain@' + SERIALNUMBER, 0) THEN
BEGIN
  ERROR('NotesInit error %1', ERP2LN.ErrorCode);
END;

ERP2LN.OpenServer('Local');

IF NOT ERP2LN.Logon('Customers.nsf') THEN
BEGIN
  ERROR('Could not open the Customers database on database Local!');
END;

ERP2LN.Form := 'CustTable2';
IF ERP2LN.Error THEN ERROR('Failed to open the form!');
```

Appendix 1. Developer reference (API)

The following terminology is used in this appendix:

Name of method or property
Syntax **<Details on parameters, data types and return value>**
Parameters ***<parameter name>*** <Description of parameter> ***<next parameter name>*** <Description of parameter>
Return Value <Description of the value returned by the property or method>
Description <General description of the method or property>
Example <Examples of usage>
Errors <Description of possible errors generated by this method or property. Usually in the format:> <ErrorCode> - <ErrorText> <ErrorCode 2> - <ErrorText 2>

ActiveHandle

Syntax

property ActiveHandle : **integer** (read/write)

Return Value

The active handle.

Description

Get/set the active handel of the current session.

Example

The following example shows how to temporary change the active handle.

```
ERP2LN.ActiveHandle := 2;
ERP2LN.OpenServer('domino1/acme');
ERP2LN.Logon('department.nsf');
ERP2LN.DocId := '0FDD'; // Select a document

ERP2LN.ActiveHandle := 3; // Change handle
ERP2LN.OpenServer('domino1/acme');
ERP2LN.Logon('department.nsf');

// Do work on another handle

ERP2LN.ActiveHandle := 2;  // Switch back
```

Errors

99 - General error - for more details see the error log.

ActiveHandle

Syntax

property ActiveHandle : **integer** (read/write)

Return Value

The active handle.

Description

Get/set the active handel of the current session.

Example

The following example shows how to temporary change the active handle.

```
ERP2LN.ActiveHandle := 2;
ERP2LN.OpenServer('Local');
ERP2LN.Logon('Customers.nsf');
ERP2LN.Form := 'CustTable';
ERP2LN.View := 'CustTable';
ERP2LN.SearchView('60000'); // Lookup customer
strSalesPersonCode := ERP2LN.FieldValue('SalesPersonCode');

ERP2LN.ActiveHandle := 3; // Change handle
ERP2LN.OpenServer('Local');
ERP2LN.Logon('Customers.nsf');
ERP2LN.Form := 'Employee';
ERP2LN.View := 'Employee';
ERP2LN.SearchView(strSalesPersonCode); // Lookup employee
MESSAGE(ERP2LN.FieldValue('Name'));

ERP2LN.ActiveHandle := 2;  // Switch back
```

Errors

99 - General error - for more details see the error log.

AddQueryField

Syntax

function AddQueryField(argFieldName: **string**) : **boolean**

Parameters

argFieldName

Name of a valid field on the currently selected form.

Return Value

True if the method succeeded; false otherwise.

Description

Specifies, that this Notes field should be included among the fields whose values is written to the comma separated file at call to *Query*.

Fields are included in the file in the order they are entered by *AddQueryField*.

Example

The following example adds two fields and executes a query.

```
ERP2LN.ClearQueryFields();
ERP2LN.AddQueryField('Number');
ERP2LN.AddQueryField('Name');
ERP2LN.Query('Nummer>''100''');
```

Errors

1 - Field is not defined on current form.
2 - No form selected.
3 - No active document or no fields on form found.
4 - Maximum number of search fields exceeded.
99 - General error - for more details see the error log.

AppendItemValue

Syntax

function AppendItemValue(argItemName: **string**; argItemValue: **string**) : **boolean**

Parameters

argItemName

Valid field name on the current form.

argItemValue

Value to append.

Return Value

True if the method succeeded; false otherwise.

Description

AppendItemValue adds the value supplied in *argItemValue* to the current values in *argItemName*.

The value must be of type Text (convert if necessary). Existing content in the field is not overwritten by this method.

AppendItemValue is often used by multi-value fields. In these cases remember to delimit the values with a delimiter character (default is ";" - see explanation for property *ListDelim*).

Example

The following example selects a document and sets the field 'Number' followed by an append to the same field.

```
ERP2LN.Form := 'CustTable';
ERP2LN.View := 'CustTable';
ERP2LN.SearchView('60000'); // Lookup customer

// Large field content (resulting in one value)
ERP2LN.ItemValue('Comments','Start of long text....'); // Set initial value
ERP2LN.AppendItemValue('Comments','Next block of text..'); // append value

// Multivalue (resulting in 3 values)
ERP2LN.ItemValue('Categories','cat1'); // Set initial value
ERP2LN.AppendItemValue('Categories',';cat2;cat3'); // append multiple values
```

Errors

1 - Field is not defined on current form.
2 - No form selected.
3 - No active document or no fields on form found.
99 - General error - for more details see the error log.

AppendTextList

Syntax

function AppendTextList(argFieldName: **string**; argFieldValue: **string**) : **boolean**

Parameters

argFieldName

Valid field name on the current form.

argFieldValue

Value to append.

Return Value

True if the method succeeded; false otherwise.

Description

This method appends the value supplied in *argFieldName* to the current values in the multi-value field supplied in *argFieldName*.

The value must be of type Text (convert if necessary). Existing content in the field is not overwritten by this method.

The difference between *AppendFieldValue* and *AppendTextList* is, that *AppendTextList* always adds a new value in a multi-value field.

Existing value are never overwritten by AppendTextList.

Example

The following example selects a document and sets the field 'Number' followed by an append to the same field.

```
ERP2LN.DocId := '0DFF'; // Select a document
ERP2LN.ItemValue('Number','Value 1');
ERP2LN.AppendTextList('Number','New multi-value');
```

Errors

1 - Field is not defined on current form.
2 - No form selected.
4 - No active document or no fields on form found.
99 - General error - for more details see the error log.

AttachFile

Syntax

function AttachFile(argFilename: **string**; argAttachmentName: **string**) : **boolean**

Parameters

argFilename

Full path to the file to attach.

argAttachmentName

Attachment name to use on the Nots document.

If an empty value is parsed the attachment name will be generated from the filename (without the path).

Return Value

True if the method succeeded; false otherwise.

Description

Attaches the file specified in *argFilename* to the current document

If *argAttachmentName* is set, it will be used to name the attachment, otherwise the *argFilename* without path is used to name the attachment.

Note: Attachment is not made to a specific rich-text field, only to the actual document).

Example

The following example attaches a file to the active document.

```
ERP2LN.AttachFile('c:\temp\import.doc', 'Report.doc');
```

Errors

2 - No form selected.
3 - No active document.
98 - argFilename cannot be read.
99 - General error - for more details see the error log.

ClearQueryFields

Syntax

procedure ClearQueryFields()

Description

Clears the list with fields to be included in result set in next call to *Query*.

Commit

Syntax

function Commit() : **string**

Return Value

On succes a 8 digit hexadecimal DocId on the commited Notes Document.

Description

Commits any changes to the currently selected document in Lotus Notes.

Data is not saved in Lotus Notes until this method has been called.

Beware that formulas on fields and form is not executed. If this is necessary call *CommitWithForm* instead.

Example

The following example updates a few items and commits the changes.

```
ERP2LN.DocId := '0DFE';
ERP2LN.SetFieldValue('Name', 'Sheet');
ERP2LN.SetFieldValue('Responsible', 'HKR');
strDocId := ERP2LN.Commit();
IF ERP2LN.Error THEN ERROR('Failed to commit changes!');
```

Errors

1 - No form selected.
2 - No data to save.
99 - General error - for more details see the error log.

CommitWithForm

Syntax

function CommitWithForm() : **string**

Return Value

On succes a 8 digit hexadecimal DocId on the commited Notes Document.

Description

This method works as *Commit* but with *CommitWithForm* all formulas on fields and the form will be executed. This method is used, if there are fields of type 'computed' on the form.

CommitWithForm can be considerably slower than Commit.

Example

The following example updates a few items and commits the changes.

```
ERP2LN.DocId := '0DFE';
ERP2LN.SetFieldValue('Name', 'Sheet');
ERP2LN.SetFieldValue('Responsible', 'HKR');
strDocId := ERP2LN.CommitWithForm();
IF ERP2LN.Error THEN ERROR('Failed to commit changes!');
```

Errors

1 - No form selected.
2 - No data to save.
99 - General error - for more details see the error log.

ComputeWithForm

Syntax

function ComputeWithForm() : **boolean**

Return Value

True if the method succeeded; false otherwise.

Description

Computes formulas on the open document.

Example

The following example shows how to compute the formulas on a document.

```
ERP2LN.DocId := '0FDD'; // Get a document
ERP2LN.FieldValue('Balance', '1000');
IF NOT ERP2LN.ComputeWithForm() THEN ERROR('Failed to compute with form!');
MESSAGE(ERP2LN.FieldValue('BalanceWithVAT'));
ERP2LN.Commit();
```

Errors

2 - No active document.
98 - Failed to compute the formulas.
99 - General error - for more details see the error log.

CreateLinkFile

Syntax

function CreateLinkFile(argFilename: **string**) : **boolean**

Parameters

argFilename

Filename on file to be created with Lotus Notes link-information (must have the extension ".url").

Return Value

True if the method succeeded; false otherwise.

Description

Creates a link-file with filename *argFilename*.

The file contains link-information on the current selected document in Lotus Notes. By double-clicking this file in Windows, the Lotus Notes client is launched and the document is opened.

Example

The following example shows how to save a link to a Lotus Notes document.

```
ERP2LN.DocId := '0DFF'; // Select a document
ERP2LN.CreateLinkFile('c:\temp\DocumentLink.url');
```

Errors

1 - No active document.
2 - Parameter argFilename must be a valid filename.
3 - Error writing the file to disk.
99 - General error - for more details see the error log.

CreateNew

Syntax

function CreateNew(argCommit: **boolean**) : **string**

Parameters

argCommit

If true the newly created document will be commited.

Return Value

If *argCommit* is true, an 8 digit hexadecimal DocId will be returned; otherwise an empty string is returned.

Description

Creates a new empty document.

Fill in values by *SetFieldValue* and save the document by calling *Commit*.

The document is created with the current form (property *Form*). If the form type in Notes is a "response-to-response document", the new form will be created as a Response document to the current document.
Similarly, a "Response document" will be created as a Response document to the main document.

If *argCommit* is true, the document will initially be commited and accessable to other processes. If *argCommit* is false it will first be availble to other processes when the method *Commit* has been called.

Example

The following example creates a new documents

```
ERP2LN.CreateNew(FALSE);
ERP2LN.SetFieldValue('Number', '125');
ERP2LN.Commit();
```

Errors

2 - No form selected.
99 - General error - for more details see the error log.

DeleteCurrentDocument

Syntax

procedure DeleteCurrentDocument()

Description

Delete the active document.

Example

The following example locates a document by DocId and deletes it.

```
  ERP2LN.Form := 'Department';
  ERP2LN.DocId := '0DFF';
  ERP2LN.DeleteCurrentDocument();
```

Errors

1 - No active document.
2 - No form selected.
99 - General error - for more details see the error log.

DeleteTempFile

Syntax

procedure DeleteTempFile()

Description

Deletes temporary files used in connection with a call to *Query*.

DetachFile

Syntax

function DetachFile(argFilename: **string**; argAttachmentName: **string**) : **boolean**

Parameters

argFilename

File to detach attachment to.

argAttachmentName

Name of attachment to detach (if blank the argFilename is used as attachment name).

Return Value

True if the method succeeded; false otherwise.

Description

Detaches *argAttachmentName* to the file specified in *argFilename*.

If *argAttachmentName* is empty the filename part of *argFilename* is used to locate the attachment.

Example

The following example attaches af document and immediately detaches it.

```
ERP2LN.AttachFile('c:\temp\report.doc', 'DayReport.doc');
ERP2LN.DetachFile('c:\temp\DayReport.doc', ''); // Assume the attachment name is
'DayReport.doc'
```

Errors

1 - Attachment not found on current document
2 - No form selected.
3 - No active document.
98 - No attachment name supplied.
99 - General error - for more details see the error log.

DocId

Syntax

property DocId : **string** (read/write)

Return Value

The currently selected document's id.

Description

Get/set the current document by document id.

The resulting comma delimited file from a call to *Query* contains DocIds in the first column.

Errors

1 - Wrong DocId (e.g. invalid ID).
2 - No form selected.
3 - Document not found.
4 - Form name missing on found document.
99 - General error - for more details see the error log.

DocLink

Syntax

property DocLink : **string** (read)

Return Value

Notes link

Description

Get a notes-link to the current selected document in Lotus Notes. This link is unique across databases, servers and replicas.

The link has the following format:

'notes://<server>/<database_replicaid>/<view_unid>/<document unid>'

Example

The following example shows how to get the a doc link to the active document.

```
ERP2LN.DocId := '0DFF'; // Select a document
strDocLink := ERP2LN.DocLink;
```

Errors

99 - General error - for more details see the error log.

DocumentLastModified

Syntax
property DocumentLastModified : **string** (read)

Return Value
A date/time value in the format: dd-mm-yyyy hh:nn:ss.

Description
Returns the last modified timestamp for the active document.

Errors
99 - General error - for more details see the error log.

Error

Syntax

property Error : **boolean** (read)

Return Value

True if the last method succeeded; false otherwise.

Description

Status of last executed function in ERP2LN.

Use *ErrorCode* and *ErrorText* to get more details related to the error

Example

The following example shows some simple error handling.

```
strDocId := ERP2LN.Commit();
IF ERP2LN.Error THEN
BEGIN
  ERROR('Failed to commit changes!\ErrorCode: ' + FORMAT(ERP2LN.ErrorCode) + '\Message: '
+ ERP2LN.ErrorText);
END
```

ErrorCode

Syntax

property ErrorCode : **integer** (read)

Return Value

The last error code; 0 if no error has occured.

Description

Last error code produced by the last executed function in ERP2LN.

Example

The following example shows how to handle an error.

```
strValue := ERP2LN.GetItemValue('Name', 'John Doe');
IF ERP2LN.Error THEN
BEGIN
  ERROR(ERP2LN.ErrorText + '(' + FORMAT(ERP2LN.ErrorCode) + ')');
END;
```

ErrorText

Syntax

property ErrorText : **string** (read)

Return Value

The last error; empty if no error has occured.

Description

Last error produced by the last executed function in ERP2LN.

Example

The following example shows how to handle an error.

```
strValue := ERP2LN.GetItemValue('Name', 'John Doe');
IF ERP2LN.Error THEN
BEGIN
  ERROR(ERP2LN.ErrorText + '(' + FORMAT(ERP2LN.ErrorCode) + ')');
END;
```

Form

Syntax

property Form : **string** (read/write)

Return Value

The active form name

Description

Get/set the active form.

Specification of form to be used for operations in Notes - e.g. data-validation. It is also possible to use form aliases.

Example

The following example...

Errors

1 - Form not found.
9 - There is no connection to the server/database (use OpenServer and Logon).
99 - General error - for more details see the error log.

FoundDocs

Syntax

property FoundDocs : **integer** (read)

Return Value

Number of documents found in last search (*Query*)

Description

Returns the number of documents found at last call to Query.

Example

The following example shows a simple query followed by a printout of found documents.

```
ERP2LN.Query('@All');
MESSAGE('Found documents: ' + Int2Str(ERP2LN.FoundDocs));
```

Errors

-99 - General error - for more details see the error log.

GetFieldValue

Syntax

function GetFieldValue(argFieldName: **string**; var argValue: **string**) : **integer**

Parameters

argFieldName

Field to get value from.

argValue

Value will be returned in this parameter.

Return Value

Length of the returned value.

Description

Query for field value on the current document. The document must be selected before the query.

If the *argFieldName* does not exist an error occurs

Example

The following example returnes a value from the current document.

```
intLength := ERP2LN.GetFieldValue('Name', strName);
IF ERP2LN.Error THEN ERROR('Error');
MESSAGE('Name: ' + strName);
```

Errors

99 - General error - for more details see the error log.

GetItemValue

Syntax

function GetItemValue(argItemName: **string**; var argValue: **string**) : **integer**

Parameters

argItemName

Item to get value from.

argValue

Value will be returned in this parameter.

Return Value

Length of the returned value.

Description

Query for item value on the current document. The document must be selected before the query.

If the *argItemName* does not exist an empty string is returned in *argValue*

Example

The following example returnes a value from the current document.

```
intLength := ERP2LN.GetItemValue('Name', strName);
IF ERP2LN.Error THEN ERROR('Error');
MESSAGE('Name: ' + strName);
```

Errors

99 - General error - for more details see the error log.

GetUnique

Syntax

function GetUnique(argItemName: **string**: argItemValue: **string**) : **string**

Parameters

argItemName

Item name (must be present on the active form)

argItemValue

Value to search for in *argItemName*.

Return Value

The document id (DocId) on the found document. Otherwise empty.

Description

This method is used for finding a document in Notes that contains *argItemValue* in item *argItemName*.

The method returns an error if the query finds more documents.

If there is a possibility that the query will result in the finding of more documents, the method *Query* should be used instead.

Remember that Notes item names are case-sensitive.

Example

The following example...

```
strDocId := ERP2LN.GetUnique('Name', 'John Doe');
IF StrLen(strDocId) > 0 THEN
BEGIN
  MESSAGE('Found document');
END;
```

Errors

0 - Document not found.
1 - argItemName is not defined on the active Notes form.
2 - Form not selected.
3 - More than 1 document found.
99 - General error - for more details see the error log.

ImportFile

Syntax

function ImportFile(argFieldName: **string**; argFilename: **string**) : **boolean**

Parameters

argFieldName

Name of the rich-text field.

argFilename

Filename to import.

Return Value

True if the method succeeded; false otherwise.

Description

Imports *argFilename* into the to the rich-text field *argFieldName*.

It is used for embedding graphics and rich-text. Can be combined with SetRichFieldValue for "ordinary" body text.

The specified file is added to the existing contents in the rich-text field.

Note that some of the graphics formats supported have some variants that are not all supported by the integration kit. It is recommended to test the compatibility of existing graphics programs by sending a mail to one's own address.

Example

The following example demonstrates how to import an image to a rich-text field.

```
ERP2LN.Form := 'CustTable';
ERP2LN.View := 'CustTable';
ERP2LN.SearchView('62000'); // Lookup customer

ERP2LN.ImportFile('CustomerLogo', 'c:\temp\logo.jpg');
ERP2LN.Commit();
```

Errors

1 - argFieldName is not defined on the Notes form.
2 - Form not selected.
3 - argFieldName is not a rich text field.
4 - No active document.
98 - File argFilename not found or cannot be read.
99 - General error - for more details see the error log.

Init

Syntax

function Init(argSerial: **string**; argOptions: **integer**) : **string**

Parameters

argSerial

The serial number obtained from the calling system (e.g. Microsoft NAV).

Depending on the calling system, the following information must be prepended to the serial number:

Calling System	Prepended string
Attain@	Microsoft Attain
Axapta@	Microsoft Axapta
Service@	Other systems

argOptions

The following options are available:

0	No options selected
99	Full debug mode

Return Value

True if the method succeeded; false otherwise.

Description

Initializes the sub-systems needed by INTOGRATE to communicate with the Lotus Notes client.

This method must always be called before any other methods are used.

Example

The following example open's a connection to a Domino server and logs on to a database.

```
CREATE(ERP2LN);
IF NOT ERP2LN.Init('Attain@' + SERIALNUMBER, 0) THEN
BEGIN
  ERROR('NotesInit error %1', ERP2LN.ErrorCode);
END;
ERP2LN.OpenServer('domino1/acme');
ERP2LN.Logon('department.nsf');
```

Errors

99 - General error - for more details see the error log.

ItemValue

Syntax

property ItemValue(argItemName: **string**) : **string** (read/write)

Return Value

The value in *argItemName*

Description

Get/sets the value of *argItemName*.

argItemName do not have to exists on the selected form.

It is possible to specify that the field is to be saved with a different type in Notes (since the field is not necessarily defined in the form).
This is done by prefixing *argItemName* with the data type.

Available data types are:

Data type	Prefix
Text	#TEXT#
Number	#NUMBER#
Date and time	#DATETIME#
Date only	#DATEONLY#
Time only	#TIMEONLY#
Text list	#TEXT_LIST#
Number list	#NUMBER_LIST#
Rich text	#RICHTEXT#

If no prefix is used the data type specified by the form will be used; otherwise it will be treated as text

Date-values must always have the format "dd-mm-yy" or "dd-mm-yyyy" - using dash as a delimiter.

The Get-method is unsupported in Microsoft NAV. Use *GetItemValue* instead.

Errors

99 - General error - for more details see the error log.

KeepUnread

Syntax

property KeepUnread : **boolean** (write)

Description

Normally, the integration kit will overwrite existing data regardless of any differences between the new field value and the old one. This means that the document will be marked as unread and LAST_MODIFIED will be changed. In certain situations, however, data may only be overwritten if the field values do differ. SetKeepUnread handles this. If it is activated the field value will be read before each write and writing in Notes will only take place if the new field value is different from the old one.

Errors

99 - General error - for more details see the error log.

KeyDelimiter

Syntax

property KeyDelimiter : **string** (write)

Description

Set the character to be used as search-field separator.

Semi-colon (;) is the default separator in connection with the transfer of multiple search values to SearchView. If this cannot be used, an alternative separator can be set by using this property.

Example

The following example show how to change the default key delimiter.

```
strQuery := Contact.Name + '@' + Contact.Address;
ERP2LN.KeyDelimiter := '@';
ERP2LN.SearchView(strQuery);
```

Errors

99 - General error - for more details see the error log.

ListDelim

Syntax

property ListDelim : **string** (write)

Description

Set the multi-value delimiter character.

To transfer values (e.g. by *SetFieldValue*) to a multi-value field each value can be separated by a delimiting character.

The default delimiter in the integration kit is ';'.

This is usefull to change when transferring values that contains semi-colon.

Example

The following example shows how to select a different delimiter.

```
ERP2LN.ListDelim := '@';
ERP2LN.SetFieldValue('ContactNotes', 'This is first; with semi colon@This is the second
note');
```

Errors

99 - General error - for more details see the error log.

Logon

Syntax

function Logon(argDatabase: **string**) : **boolean**

Parameters

argDatabase

Path to database, relative to Lotus-data directory.

Return Value

True if the method succeeded; false otherwise.

Description

Opens a session with Lotus Notes upon specification of a database name. On login Lotus Notes checks the credentials of the user.

The current set-up of the workstation (user ID, location etc.) is used when creating the connection.

When contact to a Notes server is established for the first time, the user's password is requested - unless the Notes client has been set up to share password with other programs.

Example

The following example demonstrates how to logon on to a Domino server.

```
CREATE(ERP2LN);
IF NOT ERP2LN.Init('Attain@' + SERIALNUMBER, 0) THEN
BEGIN
  ERROR('NotesInit error %1', ERP2LN.ErrorCode);
END;

ERP2LN.OpenServer('domino1/acme');
IF NOT ERP2LN.Logon('department.nsf') THEN ERROR('Error login on!');
```

Errors

9 - Server is not set. Call OpenServer first.
99 - General error - for more details see the error log.

MailAppendSignature

Syntax

property MailAppendSignature : **boolean** (write)

Description

Set if the currently active mail-signature should be appended to e-mails being send by INTOGRATE.

If you plan to import files directly into the body of the mail, then in most cases you should disable the signature.

MailAttachFile

Syntax

function MailAttachFile(argFilename: **string**; argAttachmentName: **string**) : **boolean**

Parameters

argFilename

Full path to the file to attach.

argAttachmentName

Attachment name to use on the Nots document.

If an empty value is parsed the attachment name will be generated from the filename (without the path).

Return Value

True if the method succeeded; false otherwise.

Description

Attaches the file specified in *argFilename* to the current mail

If *argAttachmentName* is set, it will be used to name the attachment, otherwise the *argFilename* without path is used to name the attachment.

Note: Attachment is not made to a specific rich-text field, only to the actual document).

Example

The following example attaches a file to the active document.

```
ERP2LN.MailSendTo('john@acme.com');
ERP2LN.MailSubject := 'January Newsletter';
ERP2LN.MailAttachFile('c:\temp\import.doc', 'Report.doc');
ERP2LN.MailSend('');
```

Errors

2 - No form selected.
3 - No active document.
98 - argFilename cannot be read.
99 - General error - for more details see the error log.

MailBCC

Syntax

procedure MailBCC(argBCC: **string**)

Parameters

argBCC

Comma-separated list of e-mail addresses (e.g. "john@acme.com" or "john@acme.com,jane@acme.com").

Description

Add an e-mail address to the BCC-field on the active e-mail.

This method can be called more than once to add several e-mail addresses.

MailBody

Syntax

function MailBody(argBody: **string**) : **boolean**

Parameters

argBody

Text to append to the body content.

Return Value

True if the method succeeded; false otherwise.

Description

Adds text to mail in the body field. By repeated calls of this method, more lines are built up.

This method must be used when importing other file formats with the method Mail-BodyFile.

It corresponds to *MailBodyText* with regard to functions but is slightly slower as it sets the value directly on the mail document.

Example

The following example sends an email with two text-lines in the body.

```
  ERP2LN.MailSendTo('john@acme.com');
  ERP2LN.MailSubject('January Newsletter');
  ERP2LN.MailBody('First line.');
  ERP2LN.MailBody('Second line.');
  ERP2LN.MailSend('');
```

Errors

1 - Error opening mail-server and/or mail-database.
99 - General error - for more details see the error log.

MailBodyText

Syntax

procedure MailBodyText(argBody: **string**)

Parameters

argBody

Text to append to the body content.

Description

Add text to the body of an e-mail.

It is similar to *MailSetBody* but it is faster since it does not require a mail document to be created (backend).

Using this method excludes the use of *MailSetBody* and *MailSetBodyFile*.

Example

The following example sends an email with to text-lines in the body.

```
ERP2LN.MailSendTo('john@acme.com');
ERP2LN.MailSetSubject('January Newsletter');
ERP2LN.MailSetBodyText('First line.');
ERP2LN.MailSetBodyText('Second line.');
ERP2LN.MailSend('');
```

Errors

99 - General error - for more details see the error log.

MailCC

Syntax

procedure MailCC(argCC: **string**)

Parameters

argCC

Comma-separated list of e-mail addresses (e.g. "john@acme.com" or "john@acme.com,jane@acme.com").

Description

Add an e-mail address to the CC-field on the active e-mail.

This method can be called more than once to add several e-mail addresses.

MailCompose

Syntax

function MailCompose(argForceBackend: **boolean**) : **string**

Parameters

argForceBackend

Create backend document.

Return Value

True if the method succeeded; false otherwise.

Description

Opens a new mail in the Lotus Notes mail client with the data set from the other mail methods.

If *argForceBackend* is TRUE, a new document will be created in Lotus Notes. This document can then be modified with the methods in the integration kit. Otherwise only a 'front-end' mail will be created.

If *argForceBackend* is FALSE, no back-end document is created.

If one or more attachments have been created on the current mail, a back-end document is always created (to store the attachments).

This method returns instantly - there will be no check whether the mail is sent or not. If a backend document has been created the UNID is returned.

Example

The following example opens a new mail in Lotus Notes with prefilled 'To' and 'Subject' fields.

```
ERP2LN.MailSendTo('john@acme.com');
ERP2LN.NotesMailSetSubject('Mail subject...');
ERP2LN.MailCompose(false);
```

Errors

99 - General error - for more details see the error log.

MailCreateLinkFile

Syntax

function MailCreateLinkFile(argFilename: **string**) : **boolean**

Parameters

argFilename

Filename on file to be created with Lotus Notes link-information (must have the extension ".url").

Return Value

True if the method succeeded; false otherwise.

Description

Creates a link-file with filename *argFilename*.

Works as *CreateLinkFile*, although with this method the file contains a link to a sent mail. The method is typically called right after *MailCompose* or *MailSend*.

Example

The following example shows how to save a link to a Lotus Notes document.

```
ERP2LN.DocId := '0DFF'; // Select a document
ERP2LN.CreateLinkFile('c:\temp\DocumentLink.url');
```

Errors

1 - No active document.
2 - Parameter argFilename must be a valid filename.
3 - Error writing the file to disk.
99 - General error - for more details see the error log.

MailGetLink

Syntax

function MailGetLink() : **string**

Return Value

Notes link

Description

This method returns a notes-link to the current prepared or sent mail in Lotus Notes. This link is unique across databases, servers and replicas.

The link has the following format:

'notes://<server>/<database_replicaid>/<view_unid>/<document unid>'

Errors

99 - General error - for more details see the error log.

MailImportBodyFile

Syntax

function MailImportBodyFile(argFilename: **string**)

Parameters

argFilename

File to import into the body content.

Return Value

True if the method succeeded; false otherwise.

Description

Imports *argFilename* into the to the body field.

It is used for embedding graphics and rich-text and can be combined with *NotesMailSetBody* to create the body content.

Note that some of the graphics formats supported have some variants that are not all supported by the integration kit. It is recommended to test the compatibility of existing graphics programs by sending a mail to one's own address.

Example

The following example sends a mail with a body content created from a file.

```
ERP2LN.MailSendTo('john@acme.com');
ERP2LN.NotesMailSetSubject('January Newsletter');
ERP2LN.NotesMailSetBody('Starting content.');
ERP2LN.NotesMailSetBodyFile('c:\temp\Chart.jpg');
ERP2LN.MailSend('');
```

Errors

99 - General error - for more details see the error log.

MailSend

Syntax

function MailSend(argBodyFilename: **string**) : **boolean**

Parameters

argBodyFilename

File to import into the body area of the mail.

The possible formats depends on your Lotus Notes version (usually the formats HTML, Text, Microsoft RTF are accepted).

Return Value

True if the method succeeded; false otherwise.

Description

Send the active mail (prepared by calling *MailSendTo*, *NotesMailSetSubject*, etc.).

If *argBodyFilename* is supplied, this file will be imported to the body content of the mail. This works the same way as when you import a file into a mail in your Lotus Notes client.

Example

The following example sends an email with 'newsletter.html' as a body.

```
ERP2LN.MailSendTo('john@acme.com');
ERP2LN.MailSubject('January Newsletter');
ERP2LN.MailSend('c:\temp\newsletter.html');
```

Errors

1 - No recipients specified. Make sure to call MailSendTo.
98 - Unable to import argBodyFilename (read error or file not found).
99 - General error - for more details see the error log.

MailSendTo

Syntax

procedure MailSendTo(argSendTo: **string**)

Parameters

argSendTo

Comma-separated list of e-mail addresses (e.g. "john@acme.com" or "john@acme.com,jane@acme.com").

Description

Add an e-mail address to the To-field on the active e-mail.

This method can be called more than once to add several e-mail addresses.

MailSubject

Syntax

procedure MailSubject(argSubject: **string**) : **string**

Parameters

argSubject

Text for the subject.

Description

Set the subject of the current mail.

Errors

99 - General error - for more details see the error log.

MoveFirst

Syntax

function MoveFirst() : **string**

Return Value

Notes Document ID of the found document.

Description

Finds and selects the first document in the last query result (Query). Returns the document's DocId.

Example

The following example shows how to iterate a query result.

```
strDocId := ERP2LN.MoveFirst();
WHILE(StrLen(strDocId)) > 0 DO
BEGIN
  ERP2LN.GetItemValue('Name', strName);
  MESSAGE(strName);

  strDocId := ERP2LN.MoveNext();
END;
```

Errors

99 - General error - for more details see the error log.

MoveLast

Syntax

function MoveLast() : **string**

Return Value

Notes Document ID of the found document.

Description

Finds and selects the last document in the last query result (Query). Returns the document's DocId.

Example

The following example shows how to iterate a query result in reverse order.

```
strDocId := ERP2LN.MoveLast();
WHILE(StrLen(strDocId)) > 0 DO
BEGIN
  ERP2LN.GetItemValue('Name', strName);
  MESSAGE(strName);

  strDocId := ERP2LN.MovePrev();
END;
```

Errors

99 - General error - for more details see the error log.

MoveNext

Syntax

function MoveLast() : **string**

Return Value

Notes Document ID of the found document.

Description

Finds and selects the next document in the last query result (Query). Returns the document's DocId.

Example

The following example shows how to iterate a query result.

```
strDocId := ERP2LN.MoveFirst();
WHILE(StrLen(strDocId)) > 0 DO
BEGIN
  ERP2LN.GetItemValue('Name', strName);
  MESSAGE(strName);

  strDocId := ERP2LN.MoveNext();
END;
```

Errors

99 - General error - for more details see the error log.

MovePrev

Syntax

function MovePrev() : **string**

Return Value

Notes Document ID of the found document.

Description

Finds and selects the previous document in the last query result (Query). Returns the document's DocId.

Example

The following example shows how to iterate a query result in reverse order.

```
strDocId := ERP2LN.MoveLast();
WHILE(StrLen(strDocId)) > 0 DO
BEGIN
  ERP2LN.GetItemValue('Name', strName);
  MESSAGE(strName);

  strDocId := ERP2LN.MovePrev();
END;
```

Errors

99 - General error - for more details see the error log.

OpenServer

Syntax

function OpenServer(argServer: **string**): **boolean**

Parameters

argServer

The name of the server to connect to.

To open a local database use "Local" as server name.

Return Value

True if the method succeeded; false otherwise.

Description

Prepare the server to connect to with *Logon*

Example

The following example open's a connection to a Domino server and logs on to a database.

```
ERP2LN.OpenServer('domino1/acme');
ERP2LN.Logon('department.nsf');
```

Errors

1 - Integration kit is not initialized - call method Init.
99 - General error - for more details see the error log.

Query

Syntax

function Query(argQuery: **string**) : **integer**

Parameters

argQuery

A Notes SELECT command, to be used for selecting Notes documents. If a blank ("") command is specified, all documents in the current selected view is selected. The word SELECT is optional.

Return Value

Number of found documents.

Description

This method is used for general queries for Notes documents. Please see your Lotus Notes documentation for details on a query statement.

If *argQuery* is empty, all documents in the database are returned.

Notes formulas can be used (e.g. "SELECT @All" or just "@All").

If a temporary file has been assigned, data is also stored in this file as comma-separated values. Remember to delete the comma-delimited file after use.

Data is also available as an in-memory result-set.

Use *MoveFirst*, *MoveNext*, *MovePrev* and *MoveLast* to navigate through this result set or read the comma file.

Example

The following example shows a simple query to the active database.

```
ERP2LN.Query('@contains(Number;"1")');
```

Errors

1 - Could not write temporary file to disk.
9 - No form selected.
99 - General error - for more details see the error log.

QueryRichTextField

Syntax

function QueryRichTextField(argFieldName: **string**) : **string**

Parameters

argFieldName

Field to get value from (must be of type rich-text)

Return Value

The content of the field (truncated if very large)

Description

Query the value from *argFieldName*.

Remember that the field name specification must be exact (Notes is case sensitive)

If a temporary file has been assigned (property *TempFile*) this will contain the text-version of *argFieldValue* on function return. Very usefull when reading large values from rich-text fields.

Example

The following example reads a rich text item from the current document.

```
ERP2LN.GetUnique('Name', 'John Doe');
strDescription := ERP2LN.QueryRichTextField('Description');
```

Errors

1 - Field not defined on active form.
2 - No form selected.
3 - No active document.
99 - General error - for more details see the error log.

QueryView

Syntax

function QueryView() : **boolean**

Return Value

True if the method succeeded; false otherwise.

Description

This method corresponds to the *Query* method, but with this method no SELECT string is used.
Instead all documents in current view are returned in a result-set and in a comma separated file (if a temporary file has been assigned).

Example

The following example queries the view 'Contacts'.

```
ERP2LN.View := 'Contacts';
ERP2LN.QueryView();
```

Errors

0 - No data found.
1 - Error creating or writing to comma separated file.
9 - View not selected.
99 - General error - for more details see the error log.

RemoveItem

Syntax

function RemoveItem(argItemName: **string**) : **boolean**

Parameters

argItemName

Name of item to delete.

Return Value

True if the method succeeded; false otherwise.

Description

Remove (deletes) *argItemName* from the active document.

The item does not have to exist as a field on the current form.

If the item is a multi-value item all values will be deleted.

Example

The following example removes the item 'MiddleName' from the active document.

```
  ERP2LN.RemoveItem('MiddleName');
```

Errors

2 - No active document.
99 - General error - for more details see the error log.

SearchFirst

Syntax

function SearchFirst : **string**

Return Value

The Notes Document ID of the found document.

Description

Returns the first document of the current view. Using Query by View makes searching much faster than searching by Query.

The View must be designed in such a way that the first sorted column MUST be of type 'text' and no sorted columns can be defined as a "constants".

Example

The following example iterates all documents in the Contacts view.

```
ERP2LN.View := 'Contacts';

strDocId := ERP2LN.SearchFirst();
WHILE(StrLen(strDocId)) > 0 DO
BEGIN
  ERP2LN.GetItemValue('Name', strName);
  MESSAGE(strName);

  strDocId := ERP2LN.SearchNext();
END;
```

Errors

0 - No data found.
1 - No view selected.
2 - No form selected.
99 - General error - for more details see the error log.

SearchNext

Syntax

function SearchNext : **string**

Return Value

The Notes Document ID of the found document.

Description

Returns the next document of the current view. Using Query by View makes searching much faster than searching by Query.

To be used typically after calls to *SearchFirst* or *SearchView.*

The View must be designed in such a way that the first column is sorted and the first column MUST be of the type 'text' and must not contain columns defined as a "constant".

Remark: if this method is used after a call to SearchView only documents matching the SearchView can be traversed.

Example

The following example iterates all documents in the Contacts view.

```
ERP2LN.View := 'Contacts';

strDocId := ERP2LN.SearchView('John Doe');
WHILE(StrLen(strDocId)) > 0 DO
BEGIN
  ERP2LN.GetItemValue('Name', strName);
  MESSAGE(strName);

  strDocId := ERP2LN.SearchNext();
END;
```

Errors

0 - No data found.
1 - No view selected.
9 - No form selected.
99 - General error - for more details see the error log.

SearchView

Syntax

function SearchView(argKeys: **string**) : **string**

Return Value

The first Notes Document ID that matches the conditions.

Description

Returns the first document in the current view which matches the search value.

Using SearchView makes searching much faster than searching by *Query* (full-text).

Should always be used in connection with lookups in views with large amounts of documents.

The view must be designed in such a way that the first column is sorted and the first column MUST be of the type 'text' and must not contain columns defined as a "constant".

If there is more than one search key, the search keys must be separated by a separator (default is semi-colon (";")). Furthermore, columns in views in Notes must match the parameter order.

Example

The following example iterates all documents with a certain key in the Contacts view.

```
ERP2LN.View := 'Contacts';

strDocId := ERP2LN.SearchFirst();
WHILE(StrLen(strDocId)) > 0 DO
BEGIN
  ERP2LN.GetItemValue('Name', strName);
  MESSAGE(strName);

  strDocId := ERP2LN.SearchNext();
END;
```

Errors

99 - General error - for more details see the error log.

TempFile

Syntax

property TempFile : **string** (read/write)

Return Value

True if the method succeeded; false otherwise.

Description

Get/set the temporary filename used by *Query*

Errors

99 - General error - for more details see the error log.

TruncateReturnString

Syntax

procedure TruncateReturnString(argMaxLength: **integer**)

Parameters

argMaxLength

The maximum number characters to return from properties and functions.

Description

When returning values from INTOGRATE there can be situations where the calling system do no support large return strings.

If this is the case this method can limit the size of returned strings.

In NAV the limit is 1024 characters.

UNID

Syntax

property UNID : **string**

Return Value

The universal id of the active document.

Description

Get/set the universal id.

When setting the UNID, the document is automatically located in the current active database.

The Universal Document Id is a 32-character string which is unique across databases.

Example

The following example locates the first record in the Contact table and selects a document in Lotus Notes.

```
ERP2LN.Form := 'ContactPerson';
IF Contact.FIND('-') THEN
BEGIN
  ERP2LN.UNID := Contact.LotusNotesUNID;
  IF NOT ERP2LN.Error THEN
  BEGIN
    MESSAGE('Found document in Lotus Notes: ' + ERP2LN.ItemValue('Subject'));
  END;
END;
```

Errors

2 - Form not set.
3 - Document with specified UNID not found.
99 - General error - for more details see the error log.

Username

Syntax

property Username: **string** (read)

Return Value

The Lotus Notes user name for the current user.

Description

Returns a fully qualified Lotus Notes user name, e.g.: "CN=John Smith/O=Intoint".

Example

The following example shows how to get the active username from Lotus Notes.

```
strUsername := ERP2LN.Username;
MESSAGE('Logged in user: ' + strUsername);
```

Errors

99 - General error - for more details see the error log.

View

Syntax

property View : **string** (write)

Return Value

True if the method succeeded; false otherwise.

Description

Set the current view

Example

The following example shows how to search a view for a given document.

```
ERP2LN.View := 'Contacts';
strDocId := ERP2LN.SearchView('John Doe');
IF StrLen(strDocId) > 0 THEN MESSAGE('Found');
```

Errors

1 - View not found.
9 - Server is not set. Call OpenServer first.
99 - General error - for more details see the error log.