SyncIT

SCRIPT SYNTAX GUIDE

WaveAccess
We Make IT Easy

HISTORY RESERVATION

Date	Version	Change	Name
28.03.2014	0.1	First Version of the document was prepared.	Roman Rudenko
20.06.2017	0.2	Second Version of the document was prepared.	Alexandra Petrova
21.02.2019	0.3	Third Version of the document was prepared.	Artem Knyazev

WaveAccess
We Make IT Easy

Table of Contents

Description

This document describes the syntax of the SyncIT scripts and capabilities.

General Script Syntax

A script for SyncIT represents an xml file which can be a valid xml file (in case of usage of root element Script as it is described in the clause or a bunch of xml elements.

The script consists of nested elements. Element is a logical document component either begins with a start-tag (for example, <context>) and ends with a matching end-tag (for example, </context>) or consists only of an empty element-tag (for example, <criteria />). The characters between the start- and end-tags, if any, are the element's content, and may contain markup, including other elements, which are called child elements.

Examples:

```
<set var="CitiesAndCountries">
     <attr name="Cities">Houston,Phoenix</attr>
     <attr name="Countries”>USA,UK,Canada</attr>
</set>
```

Names of elements, as well as names of attributes, can not contain space characters. The name should begin with a letter or an underline character. The rest of the name may contain as the same characters as well as digit characters. Attribute is a markup construct consisting of a name/value pair that exists within a start-tag or empty element-tag followed by an element name. Values of attributes should be always embedded in single or double quotes.

It is necessary to use identical types of quotes for values of attributes in the same tag (please see Cities and Countries in the example above).

Values of attributes can be a template string with expression in curly brackets (or without them). In this case a final value will be obtained by calculation of these expressions and the initial expressions will be substituted with this value. Content of elements rather will be always result of calculation, and should be specified without curly brackets. If the template string contains only one expression without other characters, the value will be the result of calculation of this expression.

Examples:

```
<set var="firstname">Joe</set>
```
- a value without expressions.

```
<if condition="contacts.Count eq 0">
```
- an expression in the attribute value.

```
<set var="calculatedvalue">{294+322}</set>
```
- an expression in the element content.

An expression may include constants and variables. A variable name is case-sensitive (please see calculatedvalue in the example above).

The expression type is selected automatically based on the expression value evaluation, but it can be converted to a necessary type by using special construction ‘as’ (available types are enumerated in the clause 3). The following examples give the same result:

```
<set var="counter">{2}</set>
<set var="counter">{2 as 'int'}</set>
```

Moreover, variable values may be a simple value, an array and a dictionary (declaration and modification of variable values are described in the clause 7).

A value of an array item can be received by specifying the array name and index of the necessary element inside square brackets. Indexes start from zero.

Example:

```
<log>First account: {account[0]}</log>
```

The random element of an array may be selected if the name of an array is specified with empty square brackets.

Example:

```
<set var="crmservers">{[crm,crm4,crm5]}</set>
<log>Random CRM server: {crmservers[]}</log>
```

A value of a dictionary item can be received by specifying the dictionary name and name of the necessary element in quotes and inside square brackets or by specifying name of the necessary element followed by the dictionary name with a dot.

Examples:

```
<log>City: {account.city}</log>
<log>City: {account["city"]}</log>
```

If the corresponding element is not found, search of a method with this name will be produced via Reflection and, if the method is found, then it will be applied to the dictionary. Otherwise, an exception will be thrown.

Example:

```
<log>Contact is found: {contacts.ContainsKey(contactid)}</log>
```

Variable types

Variables in SyncIT can be defined with one of types listed in **Table 1**.

Table 1. Variable types

Type	Description
`char`	The char keyword is used to declare a Unicode character (Unicode 16-bit character).
`char[]`	Array of char variables.
`string`	The string type represents a sequence of zero or more Unicode characters.
`string[]`	Array of string variables.
`byte`	The byte keyword is used to declare variables in range from *0* to *255* (Unsigned 8-bit integer).
`byte[]`	Array of byte variables.
`int`	The int keyword is used to declare variables in range from *-2,147,483,648* to *2,147,483,647* (Signed 32-bit integer).
`int[]`	Array of int type variables.
`long`	The long keyword is used to declare variables in range from *-9,223,372,036,854,775,808* to *9,223,372,036,854,775,807* (Signed 64-bit integer).
`long[]`	Array of long type variables.
`double`	The double keyword signifies a simple type that stores 64-bit floating-point values.
`double[]`	Array of double type variables.
`bool`	The bool keyword is used to declare variables to store the Boolean values, *true* and *false*.
`bool[]`	Array of bool variables.
`date`	The date type presents an instant in time, typically expressed as a date and time of day.
`date[]`	Array of date type variables.
`List`	Represents a strongly typed list of objects that can be accessed by index. Standard .NET List

`Dictionary`	Represents a collection of keys and values. Standard .NET Dictionary
`Object`	Generic Object, it properties can be defined in script.

Constants

There are three predefined constants in SyncIT that listed in **Table 2**.

Table 2. Constants

Type	Description
`"true"`	**True.** This constant represents a Boolean value True.
`"false"`	**False.** This constant represents a Boolean value False.
`"null"`	**Null.** This constant represents Null value

Operators

Table 3. Operators

Type	Description
`"+"`	**Addition.** This operator adds the second operand to the first operand. **Supported variable types:** int, long, double, string.
`"-"`	**Subtraction.** This operator subtracts the second operand from the first operand. **Supported variable types:** int, long, double.
`"*"`	**Multiplication.** This operator multiplies the first operand by the second operand. **Supported variable types:** int, long, double.
`"/"`	**Division.** This operator divides the first operand by the second operand. **Supported variable types:** int, long, double.
`"="`, `"=="`, `"eq"`	**Equal.** This operator returns true if the first operand is equal to the second operand.

	Supported variable types: int, long, double, string, bool, date and object.
`"!="`, `"<>"`, `"ne"`	**Not equal.** This operator returns true if the first operand is not equal to the second operand. **Supported variable types:** int, long, double, string, bool, date and object.
`">"`, `"gt"`	**Greater than.** This operator returns true if the first operand is greater than the second operand. **Supported variable types:** int, long, double, string and date.
`"<"`, `"lt"`	**Less than.** This operator returns true if the first operand is less than the second operand. **Supported variable types:** int, long, double, string and date.
`">="`, `"ge"`	**Greater or equal.** This operator returns true if the first operand is greater than or equal to the second operand. **Supported variable types:** int, long, double, string and date.
`"<="`, `"le"`	**Less or equal.** This operator returns true if the first operand is less than or equal to the second operand. **Supported variable types:** int, long, double, string and date.
`"&&"`, `"and"`	**And.** This operator performs a logical-AND of its bool operands. **Supported variable types:** bool.
`"||"`, `"or"`	**Or.** This operator performs a logical-OR of its bool operands. **Supported variable types:** bool.
`"!"`, `"not"`	**Not.** This operator is a unary operator that negates its operand. It returns true only if its operand is false. **Supported variable types:** bool.
`"[]"`, `"."`	**Index.** This operator returns an array element at the specified index or a dictionary element with the specified string key.
`"??"`	**Ternary Operation.** This operator returns the first calculated value in chain of expressions. **Example:** If variable is not null displays variable value, else display text 'MyVar was not set' `<log>{Globals['MyVar'] ?? 'MyVar was not set'}</log>`

Functions

Functions in SyncIT — are methods for set of predefined objects. There are only two predefined functions that doesn't bound to an object.

Table 4. Functions

Type	Description
`"isSet"`	**Is Set.** This function returns true if the operand has value (not null).
`"Count"`	**Count.** This function returns the number of elements in the collection.
`"Utils.NewGuid"`	**New Guid.** This function returns new random GUID.

`"Utils.Split"`	**Split.** This function identifies the substrings in a string array that are delimited by one or more characters specified in an array, then places the substrings into a specified Unicode character array.
	Example:
	`<set var="Names">Roman;Joe;Michael</set>`
	`<set var="NamesAr">{Utils.Split(Names,';')}</set>`

`"Utils.Join"`	**Join.** This function combines array values into string with specified delimiter.
	Example:
	`<set var="NamesAr">{['Roman','Joe','Michael']}</set>`
	`<set var="NamesStr">{Utils.Join(NamesAr,';')}</set>`

`"Utils.toUpper"`	**To Upper.** This function returns a copy of this string converted to uppercase. **Supported variable types:** string.
`"Utils.toLower"`	**To Lower.** This function returns a copy of this string converted to lowercase. **Supported variable types:** string.
`"Utils.NewLine"`	**New Line.** This function returns newline string ("\n").

"Utils.Now"	**Now.** This function returns a date type object that is set to the current date and time on this computer, expressed as the local time.

"Utils.Replace"	**Replace.** This function Returns a new string in which all occurrences of a specified Unicode String in the current string are replaced with another specified Unicode character or String.
	Example:
	`<set var="str1">This is an example</set>`
	`<set var="str2">{Utils.Replace(str1,'This','Here')}</set>`
	Supported variable types: string.

"FileUtils.WriteToFile"	**WriteToFile.** This function writes specified string to file, usually used for logging. Last parameter instructs to append to file. *Example:* `<set>{FileUtils.WriteToFile("c:\temp\log.txt", "Test", true)}</set>`
"FileUtils.ReadIdsFromFile"	**ReadIdsFromFile.** This function used to read predefined GUIDS from text file. GUIDS should be placed line by line. The result will be array variable. *Example:* `<set var="Ids">{FileUtils.ReadIdsFromFile(".\UserGuids.txt")}</set>`
"Xml.Load"	**Load.** This function used to loads xml file into variable *Example:* `<set xml="Xml.Load(fileName)"/>`

"Xml.Parse"	**Parse.** This function used to Load an XElement from a string that contains XML
	Example:
	`<set var="xmlSource"><![CDATA[`
	`<root>`

	<add name="abdc">Test data</add> <add name="n2">Second text data</add> </root>]]></set> <set xml="Xml.Parse(xmlSource, 1)"/>
"Math.*"	**Math.** This .NET object is used for common mathematical functions, for the list of supported methods refer to http://msdn.microsoft.com/en-us/library/system.math.aspx
"File.*"	**File.** This .NET object is used for operations with files, for the list of supported methods refer to http://msdn.microsoft.com/en-us/library/system.io.file.aspx
"Path.*"	**Path.** This .NET object is used for operations with filesystem, for the list of supported methods refer to http://msdn.microsoft.com/en-us/library/system.io.path.aspx
"Encoding.*"	**Encoding.** This .NET object is used for character and text encoding, for the list of supported methods and properties refer to http://msdn.microsoft.com/en-us/library/system.text.encoding.aspx
"TimeSpans.*"	**Timespans.** This .NET object is used for operations with time intervals. for the list of supported methods and properties refer to http://msdn.microsoft.com/en-us/library/system.timespan.aspx

Declaration of the variables

Set operation is used for declaration and modification of the variables. The name of variable can be added as **var** attribute of **set** element. However, you can skip this attribute and define variable explicitly.

Simple variable

To declare a simple variable, all that is needed is the keyword "**var**" followed by the variable name and its value.

Example:

```
<set var="accountid">{value}</set>
```

The other way to do same operation

Example:

```
<set accountid="value" />
```

Array variable

To declare an array variable simply put the values in square brackets separated by comma.

Examples:

```
<set var="crmserver">{["crm1","crm2"]}</set>
<set cities="['New York','Los Angeles','Chicago']"/>
```

Dictionary variables

To declare the dictionary variable use construction "*{new Dictionary()}*" and then set its items list using the similar way as arrays.

Example:

```
<set Address="new Dictionary()" />
<set var="Address['Street']">street</set>
<set var="Address['City']">city</set>
```

```
<set var="Address['Country']">country</set>
```

List variables

List declaration is very similar to Dictionary, use construction "*{new List()}*" and then set its variables, but without keys.

Example:

```
<set Names="new List()" />
<set var="Names[]">Michael</set>
<set var="Names[]">Joe</set>
```

Object variables

You can define any kind of object in SyncIT and there are four different methods. Let's look on examples:

Method 1:

```
<set var="Person">{new Object()}</set>
<set var="Person.Name">Michael</set>
<set var="Person.Lastname">White</set>
```

Method 2:

```
<set var="Person">
     <attr name="Name">Michael</attr>
     <attr name="Lastname">White</attr>
</set>
```

Method 3:

```
<set Person="new Object()" Person.Name="'Michael'"
Person.Lastname="'White'"/>
```

Method 4:

```
<set Person="['Name':'Michael','Lastname':'White']"/>
```

Flow Control Operations

Break

Break construction can be used to exit the cycle based on specific condition. This construction is applicable for any time of cycle.

Example:

```
<set var="testCycle">{['test1','test2','test3','test4','test5']}</set>

	<log>Start cycle.</log>
	<for var="t" in="testCycle">
		<log>{t}</log>
		<break if="t eq 'test3'"/>
	</for>
<log>End cycle.</log>


<set var="testCycle">{['test1','test2','test3','test4','test5']}</set>
	<log>Start cycle.</log>
	<for var="t" in="testCycle">
		<log>{t}</log>
		<if condition="t eq 'test3'">
			<break/>
		</if>
	</for>
<log>End cycle.</log>
```

Continue

Continue construction can be used to skip cycle logic that is located under this construction and move on to the next cycle iteration.

Example:

```
<set var="testCycle">{['test1','test2','test3','test4']}</set>

	<log>Start cycle.</log>
	<for var="t" in="testCycle">
		<continue if="t eq 'test3'"/>
		<log>{t}</log>
	</for>
<log>End cycle.</log>

	<set var="testCycle">{['test1','test2','test3','test4']}</set>
	<log>Start cycle.</log>
	<for var="t" in="testCycle">
		<if condition="t eq 'test3'">
			<continue/>
		</if>
		<log>{t}</log>
```

```
      </for>
<log>End cycle.</log>
```

For

For construction is used when a code block needs to be executed a certain amount of times. There are 2 types of usage syntax listed below.

Examples:

```
      <set crmservers="['crm','crm4','crm5']"/>
      <for var="i" from="0" to="crmservers.Count - 1" step="1">
            <log>{crmservers[i]}</log>
      </for>

      <set crmservers="['crm','crm4','crm5']"/>
      <for var="crmserver" in="crmservers">
            <log> {crmserver}</log>
      </for>
```

If and Unless

IF construction can be used when it is necessary to execute a code block only if a specified condition in the **"condition"** attribute is equal to *true*.

Example:

```
<set var="a">{10}</set>
<set var="b">{15}</set>
<if condition="a gt b">
      <log>a is greater then b</log>
</if>
<if condition="a lt b">
      <log>a is less then b</log>
</if>
<if condition="a eq b">
      <log>a is equal to b</log>
</if>
```

Unless construction is used only in conjunction with **IF** to execute code block if a condition is true and another code **IF** the condition is not true.

Any operator can use **IF** or **Unless**.

Example:

```
    <set Colors="['red','gray','yellow']"/>

    <log if="Colors[1] = 'gray'">Have a GRAY color</log>
    <log unless="Colors[1] = 'blue'">Not a BLUE color</log>

    <set var="MyColor1" if="Colors[2] = 'yellow'">Have a YELLOW color</set>
```

```
<log>{MyColor1}</log>

<set var="MyColor2" unless="Colors[1] = 'red'">Not a red color</set>
<log>{MyColor2}</log>
```

While

While construction can be used when a code block need to be executed while specified condition in the **"condition"** attribute is equal to *true*.

Example:

```
<set var="crmservers">{['crm','crm4','crm5']}</set>
<set var="counter">{0}</set>
<while condition="counter lt crmservers.Count">
   <log>{crmservers[counter]}</log>
   <set var="counter">{counter + 1}</set>
</while>
```

Then-else

THEN – ELSE construction can be used if one code block should be executed if condition is true and another code should be executed if condition is false.

Example:

```
<set var="a">{10}</set>

      <set var="b">{15}</set>
      <set var="c">{10}</set>

      <if condition="a ne b">
            <then>
                  <log>a not equal b</log>
             </then>
            <then if="b ne c">
                  <log>b not equal c</log>
            </then>
            <else>
                  <log>a equal b</log>
            </else>
            <else if="b eq c">
                  <log>b equal c</log>
            </else>
 </if>
```

Data Operations

Context

Context construction is used when it is necessary to perform a set of operation in a specific user context. Context can be used only if current user has permissions for Impersonation usage on server.

Table 5. Context constriction attributes.

Attribute	Description	Usage
`for`	A server name that a context is created for.	Required
`user`	A user Id (MS CRM) / a user E-Mail address (Exchange) that the context is used for.	Required

Example:

```
<context for="exchange" user="{userEmail}">
      <select from="exchange" entity="contact" var="contacts">
            <where>
                  <or>
                        <condition attr="crmLinkState" op="eq">0
                        </condition>
                        <condition attr="crmLinkState" op="eq">2
                        </condition>
                  </or>
                  <condition attr="crmid" op="ne"></condition>
            </where>
            <attr name="EntryId"/>
            <attr name="crmid"/>
            <attr name="crmLinkState"/>
            <attr name="iconindex"/>
            <attr name="crmOwnerId"/>
      </select>
</context>
```

Select

Select operation looks for records of specific type in accordance with the criteria and stores the retrieved data into a specified variable. The result is always an array of dictionaries.
The following tables enumerate the attributes and child elements for Read operation.

Table 6. Read operation attributes.

Attribute	Description	Usage
`from`	A server name that Read operation will run for.	Required
`entity`	A type of records that the search will perform for.	Optional
`var`	The variable name that will contain the result of search operation.	Optional
`count`	A number of records that will be returned by search	Optional
`page`	Determine from which page return records	Optional

Here in after, if “entity“ attribute has not been specified, the “contact“ entity will be used by default.

Table 7. Read operation children elements.

Element	Description	Usage
`attr`	An attribute name the found records always contain.	Required
`where`	A criteria element contains conditions set that will be used to find records to update.	Optional
`order`	Used for sorting search results by ordering.	Optional
`query`	A way to specify native query for SQL datasource	Optional

Example:

To read all names and ids of first 50 accounts where Country attribute value is equal to “US” and sort them by name attribute in descending order, you could use Select operation as follows:

```
<select from="crmserver" entity="account" var="accounts" count=”50”>
      <where>
            <condition attr="address1_country" op="eq">US</attr>
      </where>
      <attr name="name"/>
      <attr name="accountid"/>
      <order by="name" desc="true"/>
</select>
```

Select operation supports native SQL queries using query element. You can also pass parameters to the query using attr element. The entity element is required to be specified for the code to be valid, but with this method it’s not used.
Example:

```
      <select from="db" entity="tasks" var="tasks">
            <query>
                  select Tasks.Name as 'ttt', Instances.Name,
Instances.InstanceID from tasks, Instances where tasks.InstanceID =
Instances.InstanceID and Instances.Title = @title
            </query>
            <attr name="@title">Main work</attr>
      </select>
```

Select – join operation looks for records of specific type in accordance with the criteria applied to the entity itself and related entities.

Table 8. Join operation attributes.

Attribute	Description	Usage
`type`	Join type (based on used connector) For example, for CRM connector: https://docs.microsoft.com/en-us/previous-versions/dynamicscrm-2016/developers-guide/gg327702%28v%3dcrm.8%29	Required
`entity`	Related entity name.	Required
`to`	Field name of the related entity.	Required
`from`	Field name of the main entity.	Required

as	Attribute name for storing the result. Simplifies data access.	Optional

Example:

```
<select from="crm" entity="contact" var="crmContacts">
	<where>
		<condition attr="fullname" op="ex"></condition>
	</where>
	<join type='inner' entity='account' to='parentcustomerid'
from='accountid'>
		<where>
			<condition attr="name" op="ex"></condition>
		</where>
		<attr name="name" as="accountname" />
		<join entity='systemuser' to='ownerid' from='systemuserid'
as='owner'>
			<attr name="fullname" as="userfirstname"/>
		</join>
	</join>
	<attr name="firstname" />
	<attr name="lastname" as="contactlastname"/>
</select>
```

Create

Create operation creates a specified entity record with defined attributes and stores a copy of created data into the specified variable. This operation returns an array of dictionaries.

The following tables list the attributes and children elements for Create operation.

Table 9. Create operation attributes.

Attribute	Description	Usage
in	A server name that the create operation performs for.	Required
entity	A type of records that will be created.	Optional
var	The variable name that will contain a result of Create operation.	Required

Table 10. Create operation children elements.

Element	Description	Usage

attr	An attribute name and its value for a new created record.	Required

To create the contact, you could use the Create operation as follows:

```
<create in="crmserver" entity="contact" var="createdContact">
      <attr name="firstname">Joe</attr>
      <attr name="lastname">Doe</attr>
      <attr name="emailaddress1">joe.doe@mail.com</attr>
      <attr name="jobtitle">manager</attr>
      <attr name="telephone1">444-44-44</attr>
      <attr name="parentcustomerid">account:{accountid}</attr>
</create>
```

Update

Update operation updates the specified entity records corresponding with the search criteria with pre-defined attributes set. The operation returns an array of dictionaries.

Table 11. Update operation attributes.

Attribute	Description	Usage
in	A server name that the update operation performs for.	Required
entity	A type of records that will be updated.	Optional

Table 12. Update operation children elements.

Element	Description	Usage
where	A criteria element contains conditions set that will be used to find records to update.	Required
attr	An attribute name and its value for updated records.	Required

Example:

To update account with name “Adidas” and set new Primary Contact attribute, you could use the Update operation as follows:

```
<update in="crmserver" entity="account">
      <where>
            <condition attr="name" op="eq">Adidas</condition>
      </where>
```

```
      <attr name="primarycontactid">contact:{primContactId}</attr>
</update>
```

Delete

Delete operation deletes a specified entity records satisfied to search criteria.

Table 13. Delete operation attributes.

Attribute	Description	Usage
`in`	A server name that the delete operation performs for.	Required
`entity`	A type of records that will be deleted.	Optional

Table 14. The Delete operation child elements.

Element	Description	Usage
`where`	A criteria element contains conditions set that will be used to find records to delete.	Required

Example:

The following script deletes an Exchange contact by known EntryId:

```
<delete in="exchange">
      <where>
            <condition attr="EntryId" op="eq">{EntryId}</condition>
      </where>
</delete>
```

Batch

Batch operation can be used to combine create, update or delete operations into one batch to minimize the number of requests to target system.

Table 15. Batch operation attributes.

Attribute	Description	Usage
`for`	A server name that Batch operation will run for.	Required
`continueOnError`	true - continue processing the next request in the collection even if a fault has been returned from processing the current request in the collection. false - do not continue processing the next request on error.	Required
`var`	The variable name that will contain the response from target system. Response is presented only for create operation.	Required
`returnResponses`	true - return responses from each message request processed. false - do not return responses.	Optional

Example: (file for importing attached (APPENDIX 1) separetly)

```
        <set var="PageSize">{10}</set>

        <log>Script started at {Utils.Now}</log>
        <set var="csvContacts">{Csv.Read('c:/temp/contact lite test.txt',
Encoding.GetEncoding('utf-8'), ';')}</set>

        <set var="Continue">{true}</set>

        <set var="RecordsLeft">{csvContacts.Count}</set>
        <set var="Page">{0}</set>
        <log>Import started at {Utils.Now}</log>
        <log>Batch size = {PageSize}</log>
        <while condition="Continue">
              <set var="LastPage">{RecordsLeft lt PageSize}</set>
              <set var="CurrentPage" if="LastPage">{RecordsLeft}</set>
              <set var="CurrentPage" if="!LastPage">{PageSize}</set>

         <batch for="crm" continueOnError="true" var="result"
returnResponses="true">
                    <for var="iter" from="0" to="CurrentPage-1" step="1">
```

```
                        <set var="index">{(PageSize * Page) + iter}</set>
                <set var="cnt">{csvContacts[index]}</set>
                <sandbox>
                                <create in="crm" entity="contact">
                                        <attr if="cnt['lastname'].isSet and
cnt['lastname'] ne ''" name="lastname">{cnt['lastname']}</attr>
                                        <attr if="cnt['firstname'].isSet and
cnt['firstname'] ne ''" name="firstname">{cnt['firstname']}</attr>
                                        <attr if="cnt['salutation'].isSet and
cnt['salutation'] ne ''" name="salutation">{cnt['salutation']}</attr>
                                        <attr if="cnt['birthdate'].isSet and
cnt['birthdate'] ne ''" name="birthdate">{cnt['birthdate']}</attr>
                                        <attr if="cnt['slaname'].isSet and
cnt['slaname'] ne ''" name="slaname">{cnt['slaname']}</attr>
                                        <attr if="cnt['telephone1'].isSet and
cnt['telephone1'] ne ''" name="telephone1">{cnt['telephone1']}</attr>
                                        <attr if="cnt['telephone2'].isSet and
cnt['telephone2'] ne ''" name="telephone2">{cnt['telephone2']}</attr>
                                        <attr if="cnt['telephone3'].isSet and
cnt['telephone3'] ne ''" name="telephone3">{cnt['telephone3']}</attr>
                                        <attr if="cnt['websiteurl'].isSet and
cnt['websiteurl'] ne ''" name="websiteurl">{cnt['websiteurl']}</attr>
                                        <attr name="statecode">{1}</attr>
                                        <attr name="statuscode">{2}</attr>
                                </create>
                        </sandbox>
                </for>
            </batch>
            <log>{result}</log>
          <set var="RecordsLeft">{RecordsLeft - CurrentPage}</set>
          <set var="Page">{Page + 1}</set>
          <set var="Continue" if="RecordsLeft le 0">{false}</set>
        </while>
        <log>Import finished at {Utils.Now}</log>
```

Example: (Batch error handling)

```
        <if condition="result.IsFaulted">
                <log>An error occured during batch request.</log>
            <for var="resp" in="result.Responses">
                <if condition="resp.Fault.isSet">
                        <log>Error on index {resp.RequestIndex} with
message: {resp.Fault.Message}</log>
                </if>
                </for>
         </if>
```

Example: (Batch response handling for create operation)

```
        <set var="newIds">{new List()}</set>
        <for var="response" in="result.Responses">
                <if condition="!response.Fault.isSet">
```

```
                <then>
                        <set var="innerResponse">{response.Response}</set>
                        <set if="innerResponse.ResponseName eq 'Create'"
var="newIds[]">{innerResponse.id}</set>
                        <set if="innerResponse.ResponseName eq
'ExecuteTransaction'" var="newIds[]">{innerResponse.Responses[0].id}</set>
                </then>
                <else>
                        <set var="newIds[]">{null}</set>
                </else>
            </if>
        </for>
```

Transaction

Transaction operation can be used to combine create, update or delete operations into one transaction to minimize the number of requests to target system.

Table16 . Transaction operation attributes.

Attribute	Description	Usage
`for`	A server name that Batch operation will run for.	Required
`var`	The variable name that will contain the response from target system. Response is presented only for create operation.	Required
`returnResponses`	true - return responses from each message request processed. false - do not return responses.	Optional

Example: (file for importing attached separetly)

```
        <set var="PageSize">{10}</set>

        <log>Script started at {Utils.Now}</log>
        <set var="csvContacts">{Csv.Read('c:/temp/contact lite test.txt',
Encoding.GetEncoding('utf-8'), ';')}</set>

        <set var="Continue">{true}</set>

        <set var="RecordsLeft">{csvContacts.Count}</set>
```

```
        <set var="Page">{0}</set>
        <log>Import started at {Utils.Now}</log>
        <log>Batch size = {PageSize}</log>
        <while condition="Continue">
              <set var="LastPage">{RecordsLeft lt PageSize}</set>
              <set var="CurrentPage" if="LastPage">{RecordsLeft}</set>
              <set var="CurrentPage" if="!LastPage">{PageSize}</set>

          <transaction for="crm" continueOnError="true" var="result"
returnResponses="true">
                    <for var="iter" from="0" to="CurrentPage-1" step="1">
                          <set var="index">{(PageSize * Page) + iter}</set>
                <set var="cnt">{csvContacts[index]}</set>
                <sandbox>
                                    <create in="crm" entity="contact">
                                          <attr if="cnt['lastname'].isSet and
cnt['lastname'] ne ''" name="lastname">{cnt['lastname']}</attr>
                                          <attr if="cnt['firstname'].isSet and
cnt['firstname'] ne ''" name="firstname">{cnt['firstname']}</attr>
                                          <attr if="cnt['salutation'].isSet and
cnt['salutation'] ne ''" name="salutation">{cnt['salutation']}</attr>
                                          <attr if="cnt['birthdate'].isSet and
cnt['birthdate'] ne ''" name="birthdate">{cnt['birthdate']}</attr>
                                          <attr if="cnt['slaname'].isSet and
cnt['slaname'] ne ''" name="slaname">{cnt['slaname']}</attr>
                                          <attr if="cnt['telephone1'].isSet and
cnt['telephone1'] ne ''" name="telephone1">{cnt['telephone1']}</attr>
                                          <attr if="cnt['telephone2'].isSet and
cnt['telephone2'] ne ''" name="telephone2">{cnt['telephone2']}</attr>
                                          <attr if="cnt['telephone3'].isSet and
cnt['telephone3'] ne ''" name="telephone3">{cnt['telephone3']}</attr>
                                          <attr if="cnt['websiteurl'].isSet and
cnt['websiteurl'] ne ''" name="websiteurl">{cnt['websiteurl']}</attr>
                                          <attr name="statecode">{1}</attr>
                                          <attr name="statuscode">{2}</attr>
                                    </create>
                              </sandbox>
                    </for>
              </ transaction>
              <log>{result}</log>
          <set var="RecordsLeft">{RecordsLeft - CurrentPage}</set>
          <set var="Page">{Page + 1}</set>
          <set var="Continue" if="RecordsLeft le 0">{false}</set>
        </while>
         <log>Import finished at {Utils.Now}</log>
```

Search Criteria

Two operands condition operators

There are 6 types of two operands condition operators. The following table describes them in details.

Table 17. Two operands operation types.

Name	Description
`eq`	Evaluates to true if an attribute has a value that is equal to the condition value.
`ne`	Evaluates to true if an attribute has a value that is not equal to the condition value.
`lt`	Evaluates to true if the attribute has a value that is less than the condition value.
`le`	Evaluates to true if the attribute has a value that is less than or equal to the condition value.
`gt`	Evaluates to true if the attribute has a value that is greater than the condition value.
`ge`	Evaluates to true if the attribute has a value that is greater than or equal to the condition value.

Example:

The following script returns all contacts with first name “Joe”:

```
<select from="crmserver" entity="contact"
var="contacts_with_name_Joe">
      <where>
            <condition attr="firstname" op="eq">Joe</condition>
      </where>
      <attr name="contactid" />
</select>
```

Contains condition operator

Contains condition operator allow to perform text searches within string properties.Condition evaluates to true if the supplied constant value contains in the property text value.

Table 18. The Contains condition operator search patterns for CRM servers.

Pattern	Description
%text	Evaluates to true if the property text value ends with the supplied constant value.
%text%	Evaluates to true if the supplied constant value contains in the property text value.
text%	Evaluates to true if the property text value starts with the supplied constant value.

%text% is equals to search without "%" symbol.

Example:
The following script returns all contacts, which first name contains “J” and mobile phone number contains “5544”.

```
<select from="crmserver" entity="contact" var="contacts">
      <where>
            <condition attr="firstname" op="co">J</condition>
            <condition attr="mobilephone"
      op="co">5544</condition>
      </where>
      <attr name="contactid"/>
</select>
```

Contains condition ***for Exchange server*** doesn’t support the **“%”** symbol - condition evaluates to true if the supplied constant value is contained in the property text value. Search is ***case insensitive***.
Example:
The following script returns all contacts, which first name starts with “J” and mobile phone number containing “5544”.

```
<select from="exchange" entity="contact" var="contacts">
      <where>
```

```
            <condition attr="firstname" op="co">Jo</condition>
            <condition attr="mobilephone" op="co">5544</condition>
      </where>
      <attr name="EntryId"/>
</select>
```

Not operator

Not operator inverts a result of a logical operation.

Example:

The following script returns identifiers of all CRM contacts that first name is not equal to “Joe” and DoNotPhone or DoNotEmail properties are not equal to true.

```
<select from="crmserver" entity="contact" var="contacts">
      <where>
         <not>
            <and>
                  <condition attr="firstname"
            op="eq">Joe</condition>
                  <or>
                        <condition attr="donotphone" op="eq">{true}
                        </condition>
                        <condition attr="donotemail" op="eq">{true}
                        </condition>

                  </or>
            </and>
         </not>
      </where>
      <attr name="contactid"/>
</select>
```

Starts with

Starts with condition operator allow to perform text searches within string properties.Condition evaluates to true if the property text value starts with supplied constant value.

Example:

The following script returns all contacts, which first name starts with “J” and mobile phone number starts with “5544”.

```
      <select from="crmserver" entity="contact" var="contacts">
            <where>
                  <condition attr="firstname" op="sw">J</condition>
```

```
            <condition attr="mobilephone"
        op="sw">5544</condition>
        </where>
        <attr name="contactid"/>
    </select>
```

Ends with

Ends with condition operator allow to perform text searches within string properties.Condition evaluates to true if the property text value ends with supplied constant value.

Example:

The following script returns all contacts, which first name ends with “J” and mobile phone number ends with “5544”.

```
    <select from="crmserver" entity="contact" var="contacts">
        <where>
            <condition attr="firstname" op="ew">J</condition>
            <condition attr="mobilephone"
        op="ew">5544</condition>
        </where>
        <attr name="contactid"/>
    </select>
```

And / Or operators

These operators should contain two or more conditions. **And** operator evaluates to true only if all its children conditions evaluate to true. **Or** operator evaluates to true if at least one of its children conditions evaluate to true.

Example:

The following script returns all identifiers of CRM contacts which first name is equal to Joe or DoNotPhone or DoNotEmail properties are equal to true.

```
<select from="crmserver" entity="contact" var="contacts">
    <where>
        <and>
            <condition attr="firstname"
        op="eq">Joe</condition>
            <or>
                <condition attr="donotphone" op="eq">{true}
                </condition>
                <condition attr="donotemail" op="eq">{true}
```

```
                    </condition>
                </or>
            </and>
        </where>
        <attr name="contactid"/>
</select>
```

Exists condition operator

Exists condition returns true if a specified attribute exists in an entity record. It does not matter whether the property contains a non-empty value or not.

Example:
Let us imagine that there is a custom property called ShoeSize that was added to some contacts but others do not have this field.
The following script returns all contacts with this field:

```
<select from="exchange" entity="contact" var="contacts_with_ShoeSize">
        <where>
            <condition attr="ShoeSize" op="ex" />
        </where>
        <attr name="EntryId"/>
</select>
```

The following script returns all contacts without this field:

```
<select from="exchange" entity="contact" var="contacts_without_ShoeSize">
        <where>
            <not><condition attr="ShoeSize" op="ex"/></not>
        </where>
        <attr name=" EntryId "/>
</select>
```

In condition operator

In condition operator returns true if the specified attribute matches to a value in a condition values list. The following example return all identifiers of CRM contacts, which first name is Bob, Joe or Michael.

Example:

```
<select from="crmserver" entity="contact" var="contacts">
        <where>
            <condition attr="firstname" op="in">{['Bob', 'Joe',
            'Michael']}</condition>
```

```
      </where>
      <attr name="contactid" />
</select>
```

Between condition operator

Between condition operator returns true if the specified attribute value is between two values in a conditions values list. There are three ways to specify the condition's value.

Examples:
All scripts return all identifiers of CRM contacts, which were created in 2011.

```
<set var="dates">{[Utils.Now.AddDays(-100), Utils.Now.AddDays(-50)]}</set>
      <select from="crm" entity="contact" var="contacts">
            <where>
                  <condition attr="createdon"
      op="between">{[dates[0],dates[1]]}</condition>
            </where>
            <attr name="contactid" />
            <attr name="createdon" />
      </select>

      <set var="dates">{["01/01/2018", "01/01/2019"]}</set>
      <select from="crm" entity="contact" var="contacts">
            <where>
                  <condition attr="createdon"
      op="between">{[dates[0],dates[1]]}</condition>
            </where>
            <attr name="contactid" />
            <attr name="createdon" />
         </select>
```

Exception handling operations

Exception and OnError

Exception operation is used when it is necessary to throw an exception in the process of script execution.

Example:

```
<set var="crmservers">crm,crm4,crm5</set>
<for var="crmserver" in="crmservers">
   <select from="{crmserver}" entity="account" var="accounts">
            <where>
                 <condition attr="statecode"
op="eq">{0}</condition>
            </where>
            <attr name="accountid" />
      </select>
      <if condition="{accounts.Count = 0}">
            <exception>There are no active accounts in {crmserver}.
            </exception>
      </if>
</for>
```

OnError operation is used in conjunction with **Exception** for handling specific exceptions.

Example:

```
<sandbox verbose="false">
 <log>{2/0}</log>
 <onerror of="typeof System.DivideByZeroException">
  <log>This is DivideByZero!</log>
 </onerror>
</sandbox>
```

Sandbox

Sandbox operation is used when it is necessary to hide exceptions in process of script execution from user. Sandbox operation has the **"verbose"** attribute that is used for enabling or disabling errors logging (if it is left empty, then (*true*) will be used by default).

Example:
The following script looks for CRM contact "Joe Dow" and if a single instance of this contact found, then this contact is set as a primary contact of "Adidas" company. Logging is enabled for the sandbox.

```
<sandbox verbose="true">
      <select in="crmserver" entity="contact" var="contacts">
            <where>
                  <condition attr="firstname"
                  op="eq">Joe</condition>
                  <condition attr="lastname" op="eq">Dow</condition>
            </where>
            <attr name="contactid" />
      </select>

      <if condition="contacts.Count = 1">
            <update in="crmserver" entity="account">
                  <where>
                        <condition attr="name"
                  op="eq">Adidas</condition>
                  </where>
                  <attr name="primarycontactid">
                        contact:{contacts[0].contactid}
                  </attr>
            </update>
      </if>
</sandbox>
```

Structural operations

Include

Include operation is used if you want to separate some code into other script files. SyncIT will combine the main script and code from all includes on the moment of execution. This operation has the **"name"** attribute which value should be a path to existing script in regard to scripts folder plus filename without ".xml" extension.

SyncIT IDE has special order where to find script. First it tries to find specified path in a folder where a current script is situated. If the script is not found it will check Script folder in current Application folder. If after that the script is also not found it will check the standard Script folder in main SyncIT folder.

Example:

```
<include name="Includes\Parameters"/>
<include name="Includes\ReadUserDetails"/>
<include name="Includes\ReadSettings"/>
<context for="exchange" user="{User.Email}">
      <select from="exchange" var="contacts">
            <where>
                  <or>
                        <condition attr="crmLinkState" op="eq">0
                        </condition>
                        <condition attr="crmLinkState" op="eq">2
                        </condition>
                  </or>
                  <condition attr="crmid" op="ne"></condition>
            </where>
         <attr name="fileas"/>
         <attr name="EntryId"/>
         <attr name="crmid"/>
         <attr name="crmLinkState"/>
         <attr name="iconindex"/>
         <attr name="crmOwnerId"/>
      </select>
</context>

<log>Contacts to be considered: {contacts.Count}</log>
```

Call script

Call operation is used when it is necessary to execute another script (that already exists) before current script. Rather than include operation, the script will be executed in own context, and it will not have access to variables in main script. The Call operator supports passing parameters into the script. Operator searches for script in **@private** folder. There are two ways to specify call operator.

Examples:

Call script "MyScriptHelper" and pass parameter "CallSettings"

```
<script>
      <var callParameters='new Object()'/>
      <var callParameters.Param1='value1'/>

      <MyScriptHelper CallSettings="callParameters"/>
</script>

<script>
      <var callParameters='new Object()'/>
      <var callParameters.Param1='value1'/>

      <call name="MyScriptHelper" CallSettings="callParameters"/>
</script>
```

Script

Script element is used for code blocks grouping.

Example:

```
<script>
      //There should be the script body
</script>
```

Log

Log operation can be used when you need to maintain history of script executions and provide some more information to the output. It also eases process of script debugging.

Examples:

```
<script>
    <log>Result: {2+2}</log>
</script>

<script>
     <select from="crm" entity="systemuser" var="users">
          <attr name="systemuserid"/>
          <attr name="fullname"/>
     </select>

     <for var="user" in="users">
          <log>{user.systemuserid} {user.fullname}</log>
     </for>
</script>
```

Extend Functionality

You can extend script engine functionality from the script, by binding to standard .NET classes or by adding custom assemblies. The assembly should be placed in Global Assembly Cache or in the SyncIT main directory.

Example:

```
<script name="StdLib">
<!-- The following line will bind assembly to ScheduleItem
operator-->
       <set ScheduleItem="typeof
'Use4si.Infrastructure.Processing.ScheduleItem,
Use4si.Infrastructure'"/>
<!– Bind to standard .NET class-->
       <set TimeSpan="typeof System.TimeSpan"/>
<!– Bind static methods of standard .NET class -->
       <set TimeSpans="static TimeSpan"/>
</script>
```

Appendix 1

Contact Import List

Save as .txt file format

lastname;firstname;salutation;birthdate;slaname;telephone1;telephone2;telephone3;website url
Milanovic;Nikola;;;;+389 011 344 97 51;;;
Sapin;Jean-Luc;;;;546456;;;
LeVert;Albert;;;;;;;
Delarue;Jean Luc;;;;;;;
Dupond;Jean;;;;;;;
Batista;Rafael;;;;;;;
Kilic;Emrah;;;;+90 262 648 53 00;;;
Zalcberg;Ilana;;;;+55 21 3545 7124;;;
Sadaka;Mohamed;;;;+971568296165;;;
LOCHU;Philippe;;;;+33 463 05 02 97;;;
Dias Ribeiro;Daniel;;;;55 31 3248 6784;;;
Chronas;Giorgos;;;;+302106894509;;;
Martello;Marina;;;;+39 0512143791;;;
Fernandes;Althea;;;;+971044224900 ext: 906;;;
Olsen;Drew;;;;;;;
Elvin;Paul;;;;01625 238662;;;
Weckerlein;Barbara;;;;;;;
Stan;Alexandra Georgeta;;;;;;;
Streubel;Berthold;;;;+43 (0)1 40400 - 36500;;;
Pivkova-Veljanovska;Aleksandra;;;;+ 389 2 3147 783;;;
Riera;Pau;;;;;;;
Morcillo;Luis;;;;+34 651 52 31 08;;;
Dupont;Sonia;;;;+33 1 49 42 48 32;;;
Varghese;Rency;;;;;;;
Pollard;Laura;;;;(864) 388-1070;;;
Dedeurwaerdere;Fransceska;;;;051 23 71 96;;;
Kozel;Beth;;;;(314) 454-6093;;;
Etzhold;Anna;;;;+49 69 4 08 96 79 0;;;
Kooij;Dapp;;;;+32.14.64.1646;;;
Bhanushali;Pragnesh;;;;905-287-2870 x227;;;
Zavaglia;Katia;;;;;;;
BERGOUGNOUX;Anne;;;;00. 334.11.75.98.79;;;
Carcouet;Franɜois;;;;02 40 08 40 28;;;

Faurй;Julien;;;;04 76 76 63 55;;;
Vrtel;Petr;;;;;;;
;Fredy Andres Tellez;;;;0573194597772 or 05717425961 ext 116;;;
Mansour-Hendili;Lamisse;;;;;;;
Murphy;Jean;;;;+3531221 4510 / 4590;;;
Gymez;Jos Javier;;;;;;;
Tzaninis;Dimitris;;;;+302106157006;;;
Lima;Andrea;;;;55 31 2511 2200;;;
Borras;Silvia;;;;+44 1224550682;;;
Peters;Andrea;;;;+49 711 278 34901;;;
Williams;Jonathan;;;;+44 01865 225594;;;
Bento;Celeste;;;;;;;
Garuti;Anna;;;;010 3537 798;;;
Castro;Gabriela;;;;55 11 3822 2148 / 3666 2279 / 5078 8527;;;
Glotov;Andrey Sergeevich;;;;;;;
Diaz;Diego;;;;+5715700929;;;
Lindeman;Neal;;;;8573071540;;;
Badenas;Celia;;;;+34 93 227 54 00;;;
Lucarelli;Debora;;;;;;;
Kilic;Seda;;;;;;;
;Irene Pajuvuo;;;;;;;
Sorel;Nathalie;;;;;;;
Seia;Manuela;;;;+39 02 550 324 32;;;
Arnaud Gouttenoire;Estelle;;;;+41216132023;;;
Jin;Huilin;;;;;;;
Kern;Wolfgang;;;;+49 89 990 17 200;;;
Osuna;Carlos;;;;+34 976 76 56 85;;;
Cihanoglu;Cihan;;;;+90 05320514833;;;
Petrovski;Jordan;;;;+38978434837;;;
Morreau;Hans;;;;;;;
Crenshaw;Andrew;;;;;;;
Noirot;Cйline;;;;+33 3 20 44 59 62;;;
Тьmer;Zeynep;;;;+45 43 260 155;;;
Castillo;Mary;;;;51 1 513 6666;;;
Gйrard;Bйnйdicte;;;;33 (0)3 69 55 11 65;;;
Powierska-Czarny;Jolanta;;;;;;;
Cidade Batista;Marcelo;;;;+55 11 3069-6330;;;
Nagy Hafeez;Elia;;;;+201229133044;;;
Kochav;Galya;;;;972-3-9195757;;;

WaveAccess
We Make IT Easy

About

WaveAccess is a results focused software development company that provides high quality software outsourcing services to hundreds of emerging and established companies globally. We use our technical expertise to increase business efficiencies, optimize slow or unreliable systems, recover projects that have gone off track and bring ambitious ideas to life.

19
years of delivering successful outcomes for customers

350+
talented & passionate professionals

4
global R&D centers

9
industry verticals from banking to healthcare

300+
successful projects delivered and counting

96%
of our customers are repeat business

Las Vegas
headquarters

USA, UK, Denmark and Eastern Europe
sales offices

Academy Award-winning Mocha for Imagineer Systems

ISO 27001:2013 CERTIFIED

ISO 9001 CERTIFIED

Microsoft Partner Network WORLDWIDE DYNAMICS PARTNER OF THE YEAR 2009, 2010, 2011 & 2016

Silver Microsoft Partner

Microsoft Partner Network | 2011 PARTNER OF THE YEAR Microsoft Dynamics Professional Services CRM4Legal for Client Profiles Winner

Microsoft Partner | 2017 Partner of the Year Winner Business Analytics Award

Microsoft Partner | 2018 Partner of the Year Artificial Intelligence Award

WaveAccess
We Make IT Easy

If you need to develop
a similar project, please write us

hello@wave-access.com

Read more at

wave-access.com