



INTEGRATION PACK FOR DATA MANIPULATION

For Microsoft System Center Orchestrator

For System Center 2016 and 2019, you must use the 32-bit version of the integration pack, which has the name **Keverion_Integration_Pack_for_DataManipulation_4.0**

For System Center 2022 and later, you must use the 64-bit version of the integration pack, which has the name **Keverion_IP_DataManipulation_x64_4.0**

User Guide

Version 4.0

Kelverion Integration Pack for Data Manipulation

Copyright 2016 Kelverion Inc. All rights reserved.

Published: September 2022

[*Feedback*](#)

Send suggestions and comments about this document to support@kelverion.com

Contents

Kelverion Integration Pack for Data Manipulation	5
Introduction.....	5
System Requirements	5
Registering and Deploying the Integration Pack.....	5
Licensing the Integration Pack.....	7
Data Manipulation Activities.....	7
Compose Text Activity	8
Compose Text Configuration.....	8
Composing Text.....	9
Configuring Inputs for Compose Text	9
Parse Text Activity	10
Parse Text Configuration.....	10
Parsing Text.....	11
Parsing and Publishing Correlated Data	12
JSON to XML Activity	13
Converting to Valid XML	13
XML Attributes.....	13
XML Declaration.....	13
Converting JSON Arrays	14
XML to JSON Activity	14
Forcing a JSON Array	14
Apply XSLT Activity	15
Custom Date Time Formats	15
Standard Date and Time Format Specifications	15
Custom Date and Time Format Strings	15
Element Reference.....	16
XML Specification for Kelverion Integration Pack for Data Manipulation	17
<ka:DataManipulation>	17
<ka:ComposeText>.....	17
<ka:ParseText>.....	18
<ka:Attribute>.....	18
<ka:AttributeSet>.....	19
<ka:Choose>	19
<ka:Element>	20
<ka:Input>.....	21

<ka:If>	21
<ka:Message>	22
<ka:TextTemplate>	23
<ka:Otherwise>	24
<ka:Output>	25
<ka:Text>	26
<ka:When>	27
<ka:Value>	27
<ka:ValueOf>	28
<ka:Variable>	29

Kelverion Integration Pack for Data Manipulation

The Integration Pack for Data Manipulation is an add-on for Microsoft System Center Orchestrator that lets you compose and parse complicated text messages and documents.

Introduction

Workflow designers frequently encounter the need to extract and consume embedded text and data obtained from external sources that use custom formats such as XML. Similarly, to communicate with external system, it is often necessary to compose detailed textual information using data obtained from the Orchestrator data bus.

In the past, these tasks have been difficult to accomplish as you are effectively building a custom object or series of objects each time to you need to meet that specific requirement. This is not only required development level skills but also introduced duplication into your workflows and resulted in solutions that were overly complicated, fragile and difficult to support.

The goal of the Kelverion Integration Pack for Data Manipulation is to provide workflow designers with highly customizable activities that not only provide the capability to parse and compose detailed textual information but also have the ability to be inserted into any workflow and integrate seamlessly with the Orchestrator data bus.

The Kelverion Integration Pack for Data Manipulation is a powerful addition to Microsoft System Center Orchestrator that makes parsing and composing textual data fast, simple and maintainable.

System Requirements

The Integration Pack for Data Manipulation requires the following software to be installed and configured before you deploy the integration. For more information about how to install and configure System Center Orchestrator, see the respective product documentation.

Kelverion_Integration_Pack_for_DataManipulation (32-bit)

- Microsoft System Center Orchestrator 2016, 2019
- Microsoft .NET Framework 4.6.2

Kelverion_IP_DataManipulation_x64 (64-bit)

- Microsoft System Center Orchestrator 2022
- Microsoft .NET Framework 4.6.2

Registering and Deploying the Integration Pack

After you download the integration pack file, you must register it with the Orchestrator management server and then deploy it to Runbook Servers and Runbook Designers. For more information about how to install integration packs, see the [How to Install an Integration Pack](#) in the online documentation for System Center Orchestrator.

IMPORTANT: Ensure that you are deploying the correct version of the Integration Pack.

- For System Center 2016 and 2019, you must use the 32-bit version of the integration pack, which has the name **Kelverion_Integration_Pack_for_DataManipulation**
- For System Center 2022 and later, you must use the 64-bit version of the integration pack, which has the name **Kelverion_IP_DataManipulation_x64**

To register the integration pack:

1. On the management server, copy the **.OIP** file for the integration pack to a local hard drive or network share.
2. Confirm that the file is not set to **Read Only** to prevent unregistering the integration pack later.
3. Start the **Deployment Manager**.
4. In the navigation pane of the Deployment Manager, expand **Orchestrator Management Server**, right-click **Integration Packs** to select **Register IP with the Orchestrator Management Server**. The **Integration Pack Registration Wizard** opens.
5. Click **Next**.
6. In the **Select Integration Packs or Hotfixes** dialog box, click **Add**.
7. Locate the **.OIP** file that you copied locally from step 1, click **Open** and then click **Next**.
8. In the **Completing the Integration Pack Wizard** dialog box, click **Finish**.
9. On the **End User Agreement** dialog box, read the Kelverion License Terms, and then click **Accept**.
10. The **Log Entries** pane displays a confirmation message when the integration pack is successfully registered

To deploy the integration pack:

1. In the navigation pane of the **Deployment Manager**, right-click **Integration Packs**, click **Deploy IP to Runbook Server or Runbook Designer**.
2. Select the integration pack that you want to deploy, and then click **Next**.
3. Enter the name of the runbook server or computers with the Runbook Designer installed, on which you want to deploy the integration pack, click **Add**, and then click **Next**.
4. Continue to add additional runbook servers and computers running the Runbook Designer, on which you want to deploy the integration pack. Click **Next**.
5. In the **Installation Options** dialog box configure the following settings.
6. To choose a time to deploy the integration pack, select the **Schedule installation** check box, and then select the time and date from the **Perform installation** list.
7. Click one of the following:
 - a. **Stop all running runbooks before installing the integration pack** to stop all running runbooks before deploying the integration pack.
 - b. **Install the Integration Packs without stopping the running Runbooks** to install the integration pack without stopping any running runbooks.
8. Click **Next**.

9. In the **Completing Integration Pack Deployment Wizard** dialog box, Click **Finish**.
10. When the integration pack is deployed, the **Log Entries** pane displays a confirmation message.

Licensing the Integration Pack

After you register and deploy the integration pack you must provide a valid Keverion license before running any runbooks that contain activities from the integration pack

To deploy the integration pack license file to System Center Orchestrator 2019 or earlier:

1. Copy the .KAL license file to %PROGRAMFILES(X86)%\Keverion Automation\Licenses
2. Repeat for each Orchestrator Runbook Server and Runbook Designer host system.

To deploy the integration pack license file to System Center Orchestrator 2022 or later:

1. Copy the .KAL license file to %PROGRAMFILES%\Keverion Automation\Licenses
2. Repeat for each Orchestrator Runbook Server and Runbook Designer host system.

Data Manipulation Activities

This integration pack adds the **KA Data Manipulation** category to the **Activities** pane in the Client. These activities provide a flexible mechanism using custom specification files written in XML to define how to manipulate text coming from the Orchestrator data bus into another format or parsing data from a custom format and placing the parsed data onto the Orchestrator data bus.

This category contains the following activities:

- Compose Text
- Parse Text
- JSON to XML
- XML to JSON
- Apply XSLT

Compose Text Activity

The **Compose Text** activity lets you compose text (xml, html, or text) from the specified inputs.

Compose Text Configuration

Before you can use the **Compose Text** activity you must first configure an XML specification file defining the composed text.

1. Create a new XML document using your favorite text editor.
2. Add a top level [**<DataManipulation>**](#) element.
3. Inside the **<DataManipulation>** element add a [**<ComposeText>**](#) element.
 - a. Add a name attribute and assign it a unique value.
 - b. **Optional** – Add multiple **<ComposeText>** blocks inside the **<DataManipulation>** element.
4. For each input property required by your request add an [**<Input>**](#) element. Each input will be displayed in your Compose activity properties page and can be used to insert data into your request.
 - a. Add an **id** attribute and assign it an alphanumeric value. This value will uniquely identify your input.
 - b. To control how your input will appear add a **name** attribute and assign it the desired value. By default the value assigned to the **id** attribute is used.
 - c. To control whether your input is mandatory or optional add a **mandatory** attribute and assign it a value of “yes” or “no”. Inputs are mandatory by default.
 - d. To give your input a default value, add a **default** attribute and assign it the desired value.
 - e. To control the type of data represented by the input, add a **type** attribute, and assign it a value of “string”, “list”, “date”, “file”, “boolean”, “computer” or “folder”.
 - f. To control the format of date inputs add a **format** attribute and assign it the desired value. By default the system’s regional settings will be used. See [Custom Date Time Formats](#) for more information on allowed date formats.
 - g. To provide values for list inputs, add a **useValueSets** attribute and assign it a list of space delimited value set names.
5. Add a [**<TextTemplate>**](#) element. This template will be used to compose the text.
 - a. Add a **format** attribute and assign it a value of “xml”, “html” or “text”. The default format is XML.
 - b. To include or exclude the XML declaration from XML messages add an **omitXmlDeclaration** attribute and assign it a value of “no” or “yes”, respectively. XML declarations are included by default.
 - c. To control xml indentation, add an **indent** attribute and assign a value of “yes” or “no”. Xml messages are not indented by default.
 - d. To control which XML elements will be written as CDATA sections, add a **CDATAElements** attribute and assign a space delimited set of element names.

6. Define the template that will be used to build your message within the content of the **<TextTemplate>** element. See [<ka:TextTemplate>](#) for more information on building message templates.

Once you have an XML specification file prepared, you need to add a new XML Specification configuration using the Runbook Designer.

1. In the Client, click the **Options** menu, and select **KA Data Manipulation**. The **KA Data Manipulation** dialog box appears.
2. On the **Configurations** tab, click **Add** to begin the configuration setup. The **Add Configuration** dialog box appears.
3. Enter a **Name** to uniquely identify the configuration.
4. Click the ellipsis button (...) next to the **Type** box and select **XML Specification**.
5. Select your **Specification File**.
6. Click **OK** to close the configuration dialog box, and then click **Finish**.

Composing Text

When you have added one or more configurations for Data Manipulation to Microsoft System Center Orchestrator you are ready to compose text using the **Compose Text** activity.

1. Drag a **Compose Text** activity into your runbook.
2. Double click the **Compose Text** activity. You should see the Compose Text properties dialog.
3. Select a **Configuration** to define your inputs and how text is composed.
4. If your configuration has mandatory inputs these will appear in the property grid. You must provide values for all mandatory inputs.
5. If your configuration has optional inputs they may be selected by clicking **Optional Properties**. The **Add/Remove Property** dialog should appear. Select the desired inputs from the **Available** column and click **>>**. When you are finished click **OK**. The inputs you selected should appear in the property grid.
6. Click **OK**.

Configuring Inputs for Compose Text

When composing text, custom inputs can be inserted into the text using the [<ka:ValueOf>](#) element.

For example, consider a **Compose Text** used to create a text block representing an incident form. The constant text of the form is built using the **<ka:Text>** element and the appropriate values are inserted into using the **<ka:ValueOf>** element.

```

<?xml version="1.0" encoding="utf-8"?>
<ka:DataManipulation xmlns:ka="http://www.kelverion.com">
  <ka:ComposeText name="Compose Ticket">
    <ka:Input id="Impact"/>
    <ka:Input id="Priority"/>
    <ka:Input id="Urgency"/>
    <ka:Input id="Description"/>
    <ka:Input id="Opened" type="date"/>
    <ka:Input id="Submitter"/>
    <ka:TextTemplate format="text">
      <ka:Text>impact=

```

When the **Compose Text** activity runs it will generate the text using the input values and generate text similar to this:

```

impact=High&priority=High&urgency=Critical&description=Air conditioning off in
server room&opened=Wed, 26 Sep 2018 12:10:11 GMT&submitter=William Sanders

```

Parse Text Activity

The **Parse Text** activity lets you parse a text (xml, html, or text) file and place the parsed data onto the Orchestrator data bus.

Parse Text Configuration

Before you can use the **Parse Text** activity you must first configure an XML specification file defining how the text will be parsed.

1. Create a new XML document using your favorite text editor.
2. Add a top level **<DataManipulation>** element.
3. Inside the **<DataManipulation>** element add a [<ParseText>](#) element.

- a. Add a name attribute and assign it a unique value.
 - b. **Optional** – Add multiple **<ParseText>** blocks inside the **<DataManipulation>** element.
4. For each output that would be parsed from text, add an **<Output>** element.
 - a. Add a **name** attribute and assign it a value.
 - b. To include a description with your output, add a **description** attribute and assign it a value.
 - c. To control the type of published data add a **type** attribute and assign it a value of “number”, “date” or “string”. The default value is “string”.
 - d. To control how date values are parsed, add a **format** attribute, and assign an appropriate date format. For more information on date formats see [Custom Date Time Formats](#).
 - e. Add a **parser** attribute and assign it a value of “regex”, “xpath”, “split”. A value “regex” means the output will be parsed using the regular expression parser. A value of “xpath” means the output will be parsed using the XPath parser. The XPath parser is used by default. A value “split” means the output will be parsed using separated values using the user defined delimiter.
 - f. To control the regular expression parse, add a **regexOptions** attribute and assign a space delimited set of values. Allowed values include “IgnoreCase”, “IgnorePatternWhitespace”, “Multiline”, “None”, “RightToLeft” and “Singleline”.
 - g. To control how your output is published in relation to other outputs add a **correlated** attribute and assign a value of “yes” or “no”. A “yes” value means that the published data item will be published as a set with other correlated data. A “no” means the data will be published normally. Outputs are not correlated by default.
 - h. Add an expression that will be used to parse the output value from the body of the HTTP response.

Once you have an XML specification file prepared, you need to add a new XML Specification configuration using the Runbook Designer.

1. In the Client, click the **Options** menu, and select **KA Data Manipulation**. The **KA Data Manipulation** dialog box appears.
2. On the **Configurations** tab, click **Add** to begin the configuration setup. The **Add Configuration** dialog box appears.
3. Enter a **Name** to uniquely identify the configuration.
4. Click the ellipsis button (...) next to the **Type** box and select **XML Specification**.
5. Select your **Specification File**.
6. Click **OK** to close the configuration dialog box, and then click **Finish**.

Parsing Text

When you have added one or more configurations for Data Manipulation to Microsoft System Center Orchestrator you are ready to parse text using the Parse Text activity.

1. Drag a **Parse Text** activity into your runbook.

2. Double click the **Parse Text** activity. You should see the Parse Text properties dialog.
3. Select a **Configuration** to define how text will be parsed.
4. In the Properties enter a value for **Text**.
5. Click **OK**.

Parsing and Publishing Correlated Data

The published data in your Parse Text activities will frequently have mutual relationships or connections with each other. When this occurs you will want to set the value of the **correlated** attribute to "yes" to ensure that the data is published within a dataset with other correlated outputs.

For example, consider a Parse Text configured to parse employee information. It may contain the following custom outputs, which have the values of the correlated attribute, set to "yes" to indicate that they should be published in data sets.

```
<ka:Output name="Employee Count" correlated="no">count(//Employee)</ka:Output>
<ka:Output name="ID" correlated="yes">//Employee/ID</ka:Output>
<ka:Output name="First Name" correlated="yes">//Employee/FirstName</ka:Output>
<ka:Output name="Last Name" correlated="yes">//Employee/LastName</ka:Output>
<ka:Output name="Phone" correlated="yes">//Employee/Phone</ka:Output>
<ka:Output name="Email" correlated="yes">//Employee/Email</ka:Output>
<ka:Output name="Department" correlated="yes">//Employee/Department</ka:Output>
```

Provided this sample data:

```
<Employee>
  <EmployeeID>012</EmployeeID>
  <FirstName>Kevin</FirstName>
  <LastName>Walters</LastName>
  <Phone>6187852341</Phone>
  <Email>kevin.walters@acme.com</Email>
  <Department>Customer Support</Department>
</Employee>
<Employee>
  <EmployeeID>013</EmployeeID>
  <FirstName>Nancy</FirstName>
  <LastName>Haddix</LastName>
  <Phone>6187854555</Phone>
  <Email>nancy.haddix@acme.com</Email>
  <Department>Human Resources</Department>
</Employee>
```

This publishes the following correlated data to the Orchestrator data bus:

ID: 0012

First Name: Kevin

ID: 0013

First Name: Nancy

Last Name: Walters
Phone: 6187852341
Email: keven.walters@acme.com
Department: Customer Support

Last Name: Haddix
Phone: 6187854555
Email: nancy.haddix@acme.com
Department: Human Resources

JSON to XML Activity

The **JSON to XML** activity lets you convert JSON formatted text into XML. For more details about JSON specification please refer to <http://www.json.org/>. The JSON to XML Activity does not require any configuration:

1. Drag a **JSON to XML** activity into your runbook.
2. Double click the **JSON to XML** activity. You should see the JSON to XML dialog.
3. In the Properties tab enter the **JSON Text** you want to convert into XML.
4. Optionally, you can specify a **Root Element Name** to insert a top-level element in the XML.
5. Optionally, you can use the **Array Element Name** property to specify the name of the array element in the resulting XML.
6. The converted JSON is published in the **XML Text** output property.

Converting to Valid XML

Since valid XML must have a single root node, the supplied JSON must also have a single property in the root JSON object. If the JSON text has multiple root-level properties, the activity will fail. You can either edit the JSON text so that it results in proper XML, or you can use the **Root Element Name** optional property to introduce a top-level XML node.

```
{"person":{"id":"1","name":"John"},"car":{"make":"Ford"}}
```

➔ Invalid XML

```
{"info":{"person":{"id":"1","name":"John"},"car":{"make":"Ford"}}}
```

➔ Valid XML

XML Attributes

In order to convert JSON properties into XML attributes, the properties must be prefixed with '@' and they should be at start of the JSON object:

```
"person":{"@id":"1","name":"John"}
```

➔ `<person id="1"><name>John</name></person>`

XML Declaration

The XML declaration and processing attributes must be prefixed with '?':

```
{  
  "?xml":{"@version":"1.0","@encoding":"utf-8"},  
  "info":{"person":{"@id":"1","name":"John"}}  
}  
➔  
<?xml version="1.0" encoding="utf-8"?>
```

```
<info><person id="1"><name>John</name></person></info>
```

Converting JSON Arrays

The optional properties **Root Element Name** and **Array Element Name** should be specified when the **JSON Text** represents a JSON array object (it starts with '['). If these are not specified, the IP will use defaults 'RootElement' and 'ArrayElement', respectively:

```
[{"Id": "1", "Name": "Joe"}, {"Id": "2", "Name": "Jane"}]
```

➔

```
<RootElement>
  <ArrayElement>
    <Id>1</Id><Name>Joe</Name>
  </ArrayElement>
  <ArrayElement>
    <Id>2</Id><Name>Jane</Name>
  </ArrayElement>
</RootElement>
```

XML to JSON Activity

The **XML to JSON** activity lets you convert XML text into JSON text. For more details about JSON specification please refer to <http://www.json.org/>. The XML to JSON Activity does not require any configuration:

1. Drag an **XML to JSON** activity into your runbook.
2. Double click the **XML to JSON** activity. You should see the XML to JSON dialog.
3. In the Properties tab enter the **XML Text** you want to convert into JSON.
4. Optionally, you can specify to ignore the root XML node by setting **Skip Root** property to True.
5. The converted XML is published in the **JSON Text** output property.

Forcing a JSON Array

When converting from XML to JSON, by default, single elements are converted into JSON name-value properties and multiple elements at the same level are grouped into JSON arrays:

```
<info>
  <person id="1"><name>John</name></person>
</info>
```

➔

```
{"info": {"person": {"@id": "1", "name": "John"}}}
```

```
<info>
  <person id="1"><name>John</name></person>
  <person id="2"><name>Jane</name></person>
</info>
```

➔

```
{"info": {"person": [{"@id": "1", "name": "John"}, {"@id": "2", "name": "Jane"}]}}
```

To force converting a single XML element into a JSON array, use the **json:Array="true"** custom attribute:

```
<info xmlns:json="http://james.newtonking.com/projects/json">
  <person id="1" json:Array="true"><name>John</name></person>
</info>
→
{"info":{"person":[{"@id":"1","name":"John"}]}}
```

Note the XML namespace declaration is also required.

Apply XSLT Activity

The **Apply XSLT** activity lets you apply an XSL transformation on input XML. The activity does not require any configuration:

1. Drag an **Apply XSLT** activity into your runbook.
2. Double click the **Apply XSLT** activity. You should see the Apply XSL properties dialog.
3. In the **XSLT** property, enter the XSL Transform you want to apply on the input, as text.
4. In the **XML Input** property, enter the input XML, as text.
5. The transform result is published in the **XSLT Output** published data.

Custom Date Time Formats

Standard Date and Time Format Specifications

Format	Description
d	Short date pattern
D	Long date pattern
f	Long date and short time
F	Long date and long time
g	Short date and short time
G	Short date and long time.
M, m	Month day pattern
O, o	Round-trip date/time pattern.
R, r	RFC1123 date/time pattern. Always ddd, dd MMM yyyy HH:mm:ss GMT
s	ISO 8601 standard. Always yyyy-MM-ddTHH:mm:ss
t	Short time pattern
T	Long time format
u	Universal time display. Always yyyy-MM-dd HH:mm:ssZ
U	Long date and long time using universal time

Custom Date and Time Format Strings

Format	Description
d, %d	The day of the month. Single-digit days do not have a leading zero. Use %d if the format pattern is not combined with any other format specifiers.

dd	The day of the month. Single-digit days have a leading zero.
ddd	The abbreviated name of the day of the week.
dddd	The full name of the day of the week.
gg	The period or era.
h, %h	The hour in a 12 hour clock. Single digit hours do not have a leading zero. Use %h if the format pattern is not combined with other format patterns.
hh	The hour in a 12-hour clock. Single-digit hour have a leading zero.
H, %H	The hour in a 24-hour clock. Single-digit hours do not have a leading zero. Use %H if the format pattern is not combined with other format patterns.
HH	The hour in a 24-hour clock. Single-digit hours have a leading zero.
K	Local, UTC or unspecified
m, %m	The minute. Single-digit minutes do not have a leading zero. Use %m if the format pattern is not combined with other format patterns.
mm	The minute. Single-digit minutes have a leading zero.
M, %M	The numeric month. Single-digit months do not have a leading zero. Use %M if the format pattern is not combined with other format patterns.
MMMM	The full name of the month.
s, %s	The second. Single-digit seconds do not have a leading zero. Use %s if the format pattern is not combined with other format patterns.
ss	The second. Single-digit seconds have a leading zero.
t, %t	The first character in the AM/PM designator. Use %t if the format pattern is not combined with other format patterns.
tt	The AM/PM designator.
y, %y	The year without a century. If the year without a century is less than 10, the year is displayed without a leading zero. Use %y if the format pattern is not combined with other format patterns.
yy	The year without a century. If the year without the century is less than ten, the year is displayed with a leading zero.
yyy	The year in three digits. If the year is less than 100, the year is displayed with a leading zero.
yyyy	The year in four or five digits (depending on the calendar being used), including the century.
yyyyy	The year in five digits. Pads with leading zeros to get five digits.
yyyyyy	The year in six digits. Pads with leading zeroes to get six digits.
z, %z	The time zone offset. Single digit hours do not have a leading zero. Use %z if the format pattern is not combined with other format patterns.
zz	The full time zone offset. Single-digit hours have a leading zero.
:	The default time separator
/	The default date separator
\c	Where c is any character. Displays the character literally. To display the backslash character use '\\.

Element Reference

Element	Description
Attribute	Adds an XML attribute to the message body.
AttributeSet	Defines a named set of XML attributes
Choose	Used in conjunction with <When> and <Otherwise> to express multiple conditional tests.

Element	Creates an XML element node in the message body
DataManipulation	Defines the root element of the XML Specification
If	Contains a template that will be applied only if a selected condition is true.
Input	Defines an input that can be used to retrieve data from the Integration Server data bus.
Message	Writes a message to System Center Orchestrator.
TextTemplate	Defines the format and structure of the text body.
Otherwise	Specifies a default action for the <Choose> element.
Output	Defines an output that can be used to parse data from text and publish it to the Orchestrator data bus.
Text	Writes literal text in the text body
When	Specifies an action for the <choose> element.
Value	Defines a value for an input or filter.
ValueOf	Writes the value of an input variable to the text body
ValueSet	Defines a named set of values.
Variable	Declares a global variable

XML Specification for Kolverion Integration Pack for Data Manipulation

<ka:DataManipulation>

Used by: Compose Text and Parse Text

Definition and Usage

The <ka:DataManipulation> element is used to define the root element of the XML specification.

Syntax

```
<ka:DataManipulation xmlns:ka="http://www.kolverion.com"
  version="version">
  <!-- Content-->
</ka:DataManipulation>
```

Attributes

Attribute	Value	Description
version	1.0	Optional. The product version. Default is "1.0"

<ka:ComposeText>

Used by: Compose Text

Definition and Usage

The <ka:ComposeText> element is used to define the root element of a Compose Text activity.

Syntax

```
<ka:ComposeText name="name">
  <!-- Content-->
```

```
</ka:ComposeText>
```

Attributes

Attribute	Value	Description
name	name	Required. Specifies the name of this ComposeText.

<ka:ParseText>

Used by: Parse Text

Definition and Usage

The <ka:ParseText> element is used to define the root element of a Parse Text activity.

Syntax

```
<ka:ParseText name="name">
  <!-- Content -->
</ka:ParseText>
```

Attributes

Attribute	Value	Description
name	Name	Required. Specifies the name of this ParseText.

<ka:Attribute>

Used by: Compose Text

Definition and Usage

The <ka:Attribute> element is used to add attributes to XML elements.

Syntax

```
<ka:Attribute name="attributename" namespace="uri">
  <!-- Content: template -->
</ka:Attribute>
```

Attributes

Attribute	Value	Description
name	name	Required. Specifies the name of the attribute
namespace	URI	Optional. Defines the namespace URI for the attribute.

Example

```
<ka:Element name="Incident">
  <ka:Attribute name="severity">
    <ka:ValueOf select="$Severity" />
  </ka:Attribute>
</ka:Element>
```

<ka:AttributeSet>

Used by: Compose Text

Definition and Usage

The <ka:AttributeSet> element creates a named set of attributes. The AttributeSet can be applied as a whole to the text body document.

Syntax

```
<ka:AttributeSet name="name" useAttributeSets="name list">
  <!-- Content: ka:Attribute* -->
</ka:AttributeSet>
```

Attributes

Attribute	Value	Description
name	name	Required. Specifies the name of the attribute set
useAttributeSets	name list	Optional. Defines the namespace URI for the attribute.

Example

```
<ka:AttributeSet name="Critical">
  <ka:Attribute name="impact">Critical</ka:Attribute>
  <ka:Attribute name="urgency">High</ka:Attribute>
  <ka:Attribute name="priority">High</ka:Attribute>
</ka:AttributeSet>
```

<ka:Choose>

Used by: Compose Text

Definition and Usage

The <ka:Choose> element is used in conjunction with <ka:When> and <ka:Otherwise> to express multiple conditional tests. If no <ka:When> is true, the content of <ka:Otherwise> is processed. If no <ka:When> is true and no <ka:Otherwise> element is present then nothing is created. Tip: For simple conditional testing use the <ka:If> element.

Syntax

```
<ka:Choose>
  <!-- Content: (ka:When+, ka:Otherwise?) -->
</ka:Choose>
```

Example

```
<Impact>
  <ka:Choose>
    <ka:When test="$Impact='Critical'">
      <ka:Text>1</ka:Text>
    </ka:When>
    <ka:When test="$Impact='High'">
      <ka:Text>2</ka:Text>
  </ka:Choose>
</Impact>
```

```

    </ka:When>
    <ka:When test="$Impact='Medium'">
      <ka:Text>3</ka:Text>
    </ka:When>
    <ka:When test="$Impact='Low'">
      <ka:Text>4</ka:Text>
    </ka:When>
  </ka:Choose>
</Impact>

```

<ka:Element>

Used by: Compose Text

Definition and Usage

The <ka:Element> element is used to create an element in the text body.

Syntax

```

<ka:Element name="name" namespace="URI" useAttributeList="namelist">
  <!-- Content: template -->
</ka:Element>

```

Attributes

Attribute	Value	Description
name	name	Required. Specifies the name of the element to be created. The value of the name attribute can be set to an expression that is computed at run-time, like this:<ka:Element name="{ElementName}" />
namespace	URI	Optional. Specifies the namespace URI of the element. The value of the namespace attribute can be set to an expression that is computed at run-time, like this:<ka:Element name="{ElementName}" namespace="{Namespace}" />
useAttributeSets	namelist	Optional. A white space separated list of Attribute Sets containing attributes to be added to the element.

Example

```

<ka:TextTemplate format="xml">
  <ka:Element name="Incident">
    <ka:Element name="Impact">
      <ka:ValueOf select="$Impact" />
    </ka:Element>
    <ka:Element name="Urgency">
      <ka:ValueOf select="$Urgency" />
    </ka:Element>
    <ka:Element name="Priority">
      <ka:ValueOf select="$Priority" />
    </ka:Element>
    <ka:Element name="Description">
      <ka:ValueOf select="$Description" />
    </ka:Element>
  </ka:Element>
</ka:TextTemplate>

```

```

        </ka:Element>
    </ka:Element>
</ka:TextTemplate>

```

<ka:Input>

Used by: Compose Text

Definition and Usage

The <ka:Input> element defines an input that can be used to collect information from the user so that it can be formatted into the output text.

Syntax

```

<ka:Input id="id" name="name" mandatory="yes|no" default="string"
format="string" useValueSets="name list"
type="boolean|date|computer|file|folder|list|string"/>

```

Attributes

Attribute	Value	Description
id	id	Required. Specifies the unique identifier.
name	name	Optional. Specifies the name used to display the input in the Runbook Designer. Default is the value of the "id" attribute.
mandatory	yes no	Optional. "yes" specifies that the input is required. "no" specifies that the input is optional. Default is "yes".
default	string	Optional. Specifies a default value.
type	boolean date computer file folder list string	Optional. Specifies the type of data that the data represents. Some data types will include a browser when accessed from the Runbook Designer. Default is "string".
format	date format	Optional. Specifies how date/time values will be formatted. The default is the system's current regional settings.
useValueSets	name list	Optional. For list inputs, this space-delimited list of value set names is used to provide values for the list browser.

<ka:If>

Used by: Compose Text

Definition and Usage

The <ka:If> element contains a template that will be applied only if a specified condition is true.

Syntax

```
<ka:if test="expression">
  <!--Content: template -->
</ka:if>
```

Attributes

Attribute	Value	Description
test	expression	Required. Specifies the condition to be tested.

Example

```
<ka:DataManipulation xmlns:ka="http://www.kelverion.com"
  version="1.0">
  <ka:ComposeText name="Build Closed Message">
    <ka:Input id="closedAt" name="Closed Date" />
    <ka:Input id="closedBy" name="Closed By" />
    <ka:TextTemplate format="xml">
      <CloseIncident>
        <ka:If test="$closedAt='' ">
          <ka:Message terminate="yes">The 'Closed
            At' field must not be empty</ka:Message>
        </ka:If>
        <ClosedAt>
          <ka:ValueOf select="$closedAt" />
        </ClosedAt>
        <ka:If test="$closedBy='' ">
          <ka:Message terminate="yes">The
            'Closed By' field must not be empty </ka:Message>
        </ka:If>
      </CloseIncident>
    </ka:TextTemplate>
  </ka:ComposeText>
</ka:DataManipulation>
```

<ka:Message>

Used by: Compose Text

Definition and Usage

The <ka:Message> element writes a message to the text body. This element is used to report errors to System Center Orchestrator.

Syntax

```
<ka:Message terminate="yes|no">
  <!-- Content:template -->
</ka:Message>
```

Attributes

Attribute	Value	Description
terminate	yes	Optional. "yes" terminates the activity and reports the message as an

	no	error. “no” reports the message as and continues sending the message. Default is “no”.
--	----	---

Example

```
<ka:DataManipulation xmlns:ka="http://www.kelverion.com"
version="1.0">
  <ka:ComposeText name="Create Closed Message">
    <ka:Input id="closedAt" name="Closed Date"/>
    <ka:Input id="closedBy" name="Closed By"/>
    <ka:TextTemplate format="xml">
      <CloseIncident>
        <ka:If test="$closedAt=''">
          <ka:Message terminate="yes">The 'Closed By' field
            must not be empty</ka:Message>
        </ka:If>
        <ClosedAt>
          <ka:ValueOf select="$closedAt"/>
        </ClosedAt>
        <ka:If test="$closedBy=''">
          <ka:Message terminate="yes">The 'Closed By' field must
not be empty</ka:Message>
        </ka:If>
        <ClosedBy>
          <ka:ValueOf select="$closedBy"/>
        </ClosedBy>
      </CloseIncident>
    </ka:TextTemplate>
  </ka:ComposeText>
</ka:DataManipulation>
```

<ka:TextTemplate>

Used by: Compose Text

Definition and Usage

The <ka:TextTemplate> element defines the template that will be used to construct the output text.

Syntax

```
<ka:TextTemplate format="xml|text|html" cdataElements="name list"
indent="yes|no" omitXmlDeclaration="yes|no" version="1.0">
  <!-- Content:(<ka:Element>-*- , <ka:Text>-*- , template) -->
</ka:TextTemplate>
```

Attributes

Attribute	Value	Description
format	xml text html	Optional. Defines the output format. The default is XML (but if the first child of the root node is <html> and there are no preceding text nodes, then the default is HTML).

cdataElements	name list	Optional. A white-space separated list of elements whose text contents should be written as CDATA sections.
indent	yes no	Optional. "yes" indicates that the text body should be indented to its hierarchic structure. "no" indicates that the text will not be indented.
omitXmlDeclaration	yes no	Optional. "yes" specifies that the XML declaration (<?xml ...?>) should be omitted in the text body. "no" specifies that the XML declaration should be included in the text body. The default is "no".
version	string	Optional. Sets the W3C version number for the text body (only used with format="html" for format="xml").

<ka:Otherwise>

Used by: Compose Text

Definition and Usage

The <ka:Otherwise> element specifies a default action for the <ka:Choose> element. The action will take place when none of the <ka:When> conditions apply.

Syntax

```
<ka:Otherwise>
  <!-- Content: template -->
</ka:Otherwise>
```

Example

```
<Impact>
  <ka:Choose>
    <ka:When test="$Impact='Critical'">
      <ka:Text>1</ka:Text>
    </ka:When>
    <ka:When test="$Impact='High'">
      <ka:Text>2</ka:Text>
    </ka:When>
    <ka:When test="$Impact='Medium'">
      <ka:Text>3</ka:Text>
    </ka:When>
    <ka:When test="$Impact='Low'">
      <ka:Text>4</ka:Text>
    </ka:When>
    <ka:Otherwise>
      <ka:Message terminate="yes">Invalid impact value provided.
    </ka:Message>
    </ka:Otherwise>
  </ka:Choose>
</Impact>
```


<ka:Output>

Used by: Parse Text

Definition and Usage

The <ka:Output> element defines how data is parsed from input text and published to the Orchestrator data bus.

Syntax

```
<ka:Output name="string" description="string"
  parser="xpath|regex|split" type="string|number|date"
  correlated="yes|no" filtered="yes|no" useValueSets="name list">
  <!--Content: Expression -->
</ka:Output>
```

Attributes

Attribute	Value	Description
name	name	Required. Specifies the name uniquely identify the output.
correlated	yes no	Optional. "yes" specifies that the output should be published as a part of a correlated data set. "no" specifies the output should be published as an individual value. Default is "no"
description	string	Optional. Specifies a description for the output.
filtered	yes no	Optional. "yes" specifies that the output can be used to filter incoming requests for Monitor Request activities. "no" specifies that the output cannot be used for filtering. The default is "no"
parser	xpath regex split	Optional. "xpath" specifies that the XPath parser will be used to interpret the select expression. "regex" specifies that the regular expressions will be used to interpret the parse expression. "split" specifies a parser used for delimited values. The default is "xpath".
type	string number date	Optional. Specifies that published data type that will be displayed in the Orchestrator Client. The default is "string".

Example

```
<?xml version="1.0" encoding="utf-8" ?>
<ka:DataManipulation xmlns:ka="http://www.kelversion.com"
  version="1.0">
  <ka:ParseText name="Parse User Info">
    <ka:ValueSet name="provinces">
      <ka:Value>Ont</ka:Value>
      <ka:Value>BC</ka:Value>
      <ka:Value>QB</ka:Value>
    </ka:ValueSet>
  </ka:ParseText>
</ka:DataManipulation>
```

```

        <ka:Output name='Name' description='User name' type='string'
parser='xpath' correlated='yes'
filtered='yes'>//User/Name</ka:Output>
        <ka:Output name='Phone' description='User phone'
type='string' parser='xpath' correlated='yes'
filtered='yes'>//User/Phone</ka:Output>
        <ka:Output name='Provnicie' type='string' parser='xpath'
correlated='yes' filtered='yes'
useValueSets='provinces'>//User/Provnicie</ka:Output>
    </ka:ParseText>
</ka:DataManipulation>

```

<ka:Text>

Used by: Compose Text

Definition and Usage

The <ka:Text> element is used to write literal text to the text body.

Syntax

```

<ka:Text>
  <!-- Content:#PCDATA -->
</ka:Text>

```

Example

```

<ka:DataManipulation xmlns:ka="http://www.kelverion.com"
version="1.0">
  <ka:ComposeText name="Compose Ticket">
    <ka:Input id="Impact"/>
    <ka:Input id="Priority"/>
    <ka:Input id="Urgency"/>
    <ka:Input id="Description"/>
    <ka:Input id="Opened" type="date"/>
    <ka:Input id="Submitter"/>
    <ka:TextTemplate format="xml">
      <ka:Text>impact= </ka:Text>
      <ka:ValueOf select="$Impact"/>
      <ka:Text>&priority= </ka:Text>
      <ka:ValueOf select="$Priority"/>
      <ka:Text>&urgency= </ka:Text>
      <ka:ValueOf select="$Urgency"/>
      <ka:Text>&description= </ka:Text>
      <ka:ValueOf select="$Description"/>
      <ka:Text>&opened= </ka:Text>
      <ka:ValueOf select="$Opened"/>
      <ka:Text>&submitter= </ka:Text>
      <ka:ValueOf select="$Submitter"/>
    </ka:TextTemplate>
  </ka:ComposeText>
</ka:DataManipulation>

```

<ka:When>

Used by: Compose Text

Definition and Usage

The <ka:When> element is used to specify an action for the ka:Choose element. The <ka:When> element evaluates an expression and if it returns true then an action is performed.

Note: The <ka:When> element is used in conjunction with <ka:Choose> and <ka:Otherwise> to express multiple conditional tests.

Syntax

```
<ka:When test="boolean expression">
  <!-- Content: template -->
</ka:When>
```

Attributes

Attribute	Value	Description
test	boolean expression	Required. Specifies a boolean expression to be tested.

Example

```
<Impact>
  <ka:Choose>
    <ka:When test="$Impact='Critical'">
      <ka:Text>1</ka:Text>
    </ka:When>
    <ka:When test="$Impact='High'">
      <ka:Text>2</ka:Text>
    </ka:When>
    <ka:When test="$Impact='Medium'">
      <ka:Text>3</ka:Text>
    </ka:When>
    <ka:When test="$Impact='Low'">
      <ka:Text>4</ka:Text>
    </ka:When>
  </ka:Choose>
</Impact>
```

<ka:Value>

Used by: Compose Text and Parse Text

Definition and Usage

Defines a list browser value for an input for filter.

Syntax

```
<ka:Value>
  <!--Content:string-->
</ka:Value>
```

Example

```
<ka:DataManipulation xmlns:ka="http://www.kelverion.com"
version="1.0">
  <ka:ParseText name="Parse Ticket">
    <ka:ValueSet name="Impact">
      <ka:Value>High</ka:Value>
      <ka:Value>Medium</ka:Value>
      <ka:Value>Low</ka:Value>
    </ka:ValueSet>
    <ka:ValueSet name="Urgency">
      <ka:Value>High</ka:Value>
      <ka:Value>Medium</ka:Value>
      <ka:Value>Low</ka:Value>
    </ka:ValueSet>
    <ka:ValueSet name="Priority">
      <ka:Value>Critical</ka:Value>
      <ka:Value>High</ka:Value>
      <ka:Value>Moderate</ka:Value>
      <ka:Value>Low</ka:Value>
    </ka:ValueSet>
    <ka:Output name='Priority' type='string' parser='xpath'
correlated='yes' filtered='yes'
useValueSets='Priority'>//Ticket/Priority</ka:Output>
  </ka:ParseText>
</ka:DataManipulation>
```

<ka:ValueOf>

Used by: Compose Text

Definition and Usage

The <ka:ValueOf> element inserts the value of a specified input.

Syntax

```
<ka:ValueOf select="expression"/>
```

Attributes

Attribute	Value	Description
select	expression	Required. An XPath expression that input to insert.

Example

```
<ka:DataManipulation xmlns:ka="http://www.kelverion.com"
version="1.0">
  <ka:ComposeText name="Compose Ticket">
```

```

<ka:Input id="Impact"/>
<ka:Input id="Urgency"/>
<ka:Input id="Priority"/>
<ka:Input id="Description"/>
<ka:Input id="Submitter"/>
<ka:TextTemplate format="xml">
  <Incident>
    <Impact>
      <ka:ValueOf select="$Impact"/>
    </Impact>
    <Urgency>
      <ka:ValueOf select="$Urgency"/>
    </Urgency>
    <Priority>
      <ka:ValueOf select="$Priority"/>
    </Priority>
    <Description>
      <ka:ValueOf select="$Description"/>
    </Description>
    <Submitter>
      <ka:ValueOf select="$Submitter"/>
    </Submitter>
  </Incident>
</ka:TextTemplate>
</ka:ComposeText>
</ka:DataManipulation>

```

<ka:Variable>

Used by: Compose Text

Definition and Usage

The <ka:Variable> declares a global variable that can be used to insert a constant value into the text body.

Syntax

```

<ka:Variable name="name" select="expression">
  <!-- Content: template -->
</ka:Variable>

```

Attributes

Attribute	Value	Description
name	Name	Required. Specifies the name of the variable.
select	expression	Optional. Defines the value of the variable.