



G/On Customization Reference

Reference information for customization of G/On

G/On version: 5.3

Document revision: 0.93

About this document

This document gives an in-depth description of the options there exist for customization of G/On Management.

If you do not find the information you need in this document, you may want to look in the other documents in the G/On software documentation suite:

- G/On User Guide – Getting started – Fedora
- G/On User Guide – Getting started – Windows XP
- G/On User Guide – Getting started – Windows Vista
- G/On User Guide – Getting started – Windows 7
- G/On User Guide – Getting started – Mac
- G/On User Reference
- Getting started with G/On Set-up and Configuration
- Getting started with G/On Management
- G/On Set-up and Configuration Reference
- G/On Management Reference
- G/On Customization Reference

© Giritech A/S, 2009
Spotorno Allé 12, 2.
2630 Taastrup
Denmark
Phone +45 70.277.262

Legal Notice

Giritech reserves the right to change the information contained in this document without prior notice. Giritech® and G/On™ are trademarks and registered trademarks of Giritech A/S. Giritech A/S is a privately held company registered in Denmark. Giritech's core intellectual property currently includes the patented systems and methods known as EMCADS™. Other product names and brands used herein are the sole property of their owners. Unauthorized copying, editing, and distribution of this document is prohibited.

Contents

About this document.....	2
Introduction.....	3
Packages and Package Collection Specifications.....	4
Creating new or revised package and collection specifications.....	4
Package Specifications (gpmdef.xml).....	4
Package Collection Specifications (gpmcdef.xml).....	5
XML schema.....	7
Menu Action templates.....	8
Creating new or revised templates.....	8
Syntax.....	8
Field specification.....	13
Variables.....	16

Introduction

In G/On 5 it is possible to include support for new kinds of applications, so they can work through G/On:

1. The software (e.g. the application client) that must be run on the client PCs can be packaged and easily distributed to the client PCs. The format for defining such client software *packages* is described in the following.
2. When preparing a token before giving it to a user, it is possible to put a whole *collection* of software packages on the token. The format for defining special collections with the packages of your choice, is also described below.
3. When introducing a new application, it is necessary to define how this should be started: Which client side program to start, which parameters to supply, and which communication connections to allow to the application server, through G/On. Usually, some of this information will be fixed for all uses of the application, while some depends on the specifics of the actual IT infrastructure. Below, we will introduce a format for describing so-called Menu Item Templates. The templates capture both the fixed and the parameterized parts of the application start-up information, and each template appears in the G/On Management Client as a Menu Action Wizard.

Packages and Package Collection Specifications

This section describes the syntax and semantics of Packages Specifications and Package Collection Specifications. Packages Specifications are XML files, which define individual client software packages containing e.g. the files of a G/On client or an application client, or some other set of files. Packages Collection Specifications are also XML files; they define often used “bundles” of packages which can be put on a token when initializing it. First we give a description on where the specifications are found and how to create, change or delete them. After that, follow two subsections with descriptions of the details of Package Specifications and Package Collection Specifications,

Creating new or revised package and collection specifications

Package and Package Collection Specifications are located in the folders:

```
.\gon_server_admin_service\win\gpm\gpmdefs  
.\gon_server_admin_service\win\gpm\gpmcdefs
```

The folder already contains pre-defined specifications. If you want to change these, you can do so, but we recommend that you create a copy and edit that instead.

You can create new specification. The easiest way to make a new specification is simply to copy one of the existing ones, and change it. Use, e.g. Wordpad for editing, or an editor that is good for editing XML. Note that all packages and package collections must have a unique name (see the following subsections).

After making changes to package or collection specifications, always click the button: “Generate” in the section: “Software Package (GPM) generation”, in the G/On Configuration program. This will generate the actual packages and recompute cached information about the collections. The results are placed in the folder:

```
.\gon_server_admin_service\win\gpm\gpms
```

To remove a package or collection, remove the appropriate XML file, and also remove any unwanted gpm package file from the folder:

```
.\gon_server_admin_service\win\gpm\gpms
```

Note: After changes, you may need to restart G/On Management, in order to use new/changed package collections. And you will need to restart G/On clients, in order that new/changed packages can be installed/updated, using the G/Update menu actions.

Package Specifications (gpmdef.xml)

This section describes the package specification format **gpmdef**. From a package specification, a package(gpm-file) can be generated.

The package specification contains meta information about a package e.g. version and description, and it contains the selection of which files should be included, and where they should be placed, when the package is installed.

The following example is the package specification of the gon_client package for the mac platform:

```

<?xml version="1.0" encoding="UTF-8"?>
<gpmdef>
  <header name="gon_client">
    <version main="5" minor="4" bugfix="0" build_id="0" />
    <version_release main="1"/>
    <arch>mac</arch>
    <description lang="en" summary="G/On Client">This package contains the G/On Client for the Mac platform.</description>
  </header>

  <filedefs_ro>
    <include source="{gpm_build_root}/gon_client/mac" dest="/gon_client/mac"/>
    <include source="{gpm_build_root}/gon_client/mac/gon_client.app/Contents/Resources/shortcut/G-On Mac" dest="/G-On Mac"/>
  </filedefs_ro>
</gpmdef>

```

The main element *gpmdef* contains a *header* element which holds the meta information. The *version* element is used for holding the version of the included software or included files, and the *version_release* element is the version for the package specification itself. The *version_release* should be increased if something is changed in the package e.g. an extra file is added, or the description is changed. The combined version number of the *version* element and the *version_release* element constitutes the full version number of the generated package, and is used by the update functionality to detect packages that should be updated.

The main element *gpmdef* can contain a *filedefs_ro* element and/or a *filedefs_rw* element. Both elements contains the specification of which files to include, and where to put them when the package is installed. The *filedef_ro* specify the files to be installed to the Read-Only part, and *filedef_rw* specify the files to be installed to the Read-Write part of the destination token. How the Read-Only part and Read-Write part are interpreted depends on the token. Notice that the Read-Only part and Read-Write part could be the same, e.g. on a SoftToken. In order not to introduce errors, this means that the destination of files included by the *filedefs_ro* element and *filedefs_rw* must not overlap. The *filedefs_ro* element and *filedefs_rw* element can contain a number of *include* and/or *ignore* elements. The *include* element can be used to include the content of a folder, but it can also be used to include a single file, see the example above. The *ignore* element excludes files that would otherwise be included.

The main element *gpmdef* can contain a *dependency* element which defines the relation to other packages. The following example specify that the package 'my_package' is to be installed, and that the 'old_package' should be removed.

```

...
<dependency>
  <requires>
    <packageref name="my_package" arch="win"/>
  </requires>
  <obsolutes>
    <packageref name="old_package" arch="win"/>
  </obsolutes>
</dependency>
...

```

Package Collection Specifications (gpmcdef.xml)

This section describes the package collection format *gpmcdef*, in order to enable you to create your own collections. A Package Collection is an XML definition of a collection of packages, which can be used in the “Token Software Management” view in G/On Management.

A number of package collections are included in the standard G/On system. Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<gpmcdef>
  <header name="all_clients">
    <description lang="en" summary="GOn Client for all platforms">Collection of GOn Client packages all platforms.</description>
  </header>
  <packages>
    <packageref name="gon_client" arch="win"/>
    <packageref name="gon_client" arch="mac"/>
    <packageref name="gon_client" arch="linux"/>
  </packages>
</gpmcdef>
```

The main element *gpmcdef* consists of two child elements; *header* and *packages*.

The *header* element has an attribute *name*, which is the unique identifier for the collection. The child element, *description*, contains the long description, which will appear in the “description” section in “Token Software Management” when a collection is chosen. The content of the attribute *summary* is the one that will be shown in the list of collections in “Token Software Management”. The attribute *lang* is not used in this version. In the future there may be a number of descriptions in different languages, but in the current version the first *description* child element is always used.

The *packages* element is the one containing references to the packages, which should be included in the collection. It contains a number of *packageref* child elements, each of which must have a *name* and *arch* attribute. The *name* and *arch* attributes refer to the same entries for the package in question. Here is an example of how a package definition looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<gpmcdef>
  <header name="gon_client">
    <version main="5" minor="3" bugfix="0" build_id="8" />
    <version_release main="1"/>
    <arch>win</arch>
    <description lang="en" summary="G/On Client">This package contains the G/On Client for the Windows platform.</description>
  </header>
  ...
</gpmcdef>
```

In this example, notice the header element with a *name* attribute and an *arch* child element. These are the values, which should be used if wanting to include this package in a collection.

XML schema

This is the XML Schema for Package Collections:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="gpmcdef">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="header">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="description">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="lang" type="xs:string" use="required" />
                      <xs:attribute name="summary" type="xs:string" use="required" />
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="packages">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="packageref">
          <xs:complexType>
            <xs:attribute name="name" type="xs:string" use="required" />
            <xs:attribute name="arch" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Menu Action templates

This section describes the syntax and semantics of Menu Action templates. Menu Action templates are XML files, which define templates for the Menu action Wizard. First we give a description on where templates are found and how to create, change or delete them. After that, a description of the template syntax is given. Finally lists of fields and variables, which can be used in templates, is presented along with a short description of their meaning. There is more information on the meaning of fields in the G/On Management Reference.

Creating new or revised templates

Templates are located in the `.\gon_server_admin_service\win\templates` folder. The folder already contains pre-defined templates. If you want to change these templates you can do so, but we recommend that you create a copy and edit that instead.

You can create new templates. The easiest way to make a new template is simply to copy one of the existing ones, and change it. Use, e.g. Wordpad for editing, or an editor that is good for editing XML. Note that all templates must have a unique name (see Syntax). If two templates have the same name only one of them will appear in the management client.

To remove a wizard template, simply remove the appropriate XML file from templates folder.

After making changes to the templates folder, restart the management client, to use the new set of wizard templates.

Syntax

In this section we describe the elements that make up a Menu Action template. We will use examples to show the various elements and attributes which can be used. The XML schema for templates is provided in the end of this section.

Basic features

The template consists of a header element `ConfigurationTemplate`, which contains a number of field elements and possibly a description element. The `ConfigurationTemplate` element has two parameters:

- `name` : The template name. This must be unique.
- `title` : The template title. This is the title which will appear in the administration client

The `ConfigurationTemplate` element can contain a `description` child element. The content of this element will appear next to the template title on the wizard chooser page.

The central part of the template are the fields, which are also children of the `con` element.

There must always be at least one `field` element in the template. Each `field` has a mandatory attribute `field_type`, which can be either `edit`, `hidden` OR `custom_template`. Fields of type `edit`, will appear as input fields in the Menu Action wizard in the order specified in the template. Fields with attribute `advanced` set, will appear in the “Advanced” section of the wizard. Fields of type `hidden` will not be shown. Fields of type `custom_template` are described in the Custom fields section below. Each field must contain a `name` element. The content of this should be the name of a Menu Action field. Please refer to the Field specification section, for a list of the fields available.

The following elements are allowed in a field:

- `name` : Unique field name referring to Menu Action field (except for custom fields).
- `default_value` : Initial value for the field.
- `title` : Field title shown in the client. If not specified, a default title is used.
- `tooltip` : Text shown as tool tip in the administration client.
- `description` : A description of the field. Not used at the moment.
- `selection` : Drop-down option – see below.
- `action` : Add action button next to field – see below.

And these are the possible attributes for a field:

- `field_type` : Can be `edit`, `hidden` OR `custom_template`
- `advanced` : Boolean. True if field should appear in advanced section
- `max_length` : Integer. Maximum number of characters which can be entered.
- `secret` : Boolean. If set the typed value will not be shown (password field).
- `read_only`: Boolean. Should field be read-only
- `mandatory` : Boolean. Should this field always be set.

Example of a simple template:

```
<?xml version="1.0" encoding="utf-8" ?>
<ConfigurationTemplate name="example" title="Template Example" xmlns="http://giritech.com/admin_ws">
  <description>Sample template</description>

  <field field_type="edit">
    <name>label</name>
  </field>

  <field field_type="edit">
    <title>Server</title>
    <name>portforward.0.server_host</name>
  </field>

  <field field_type="hidden">
    <name>launch_type</name>
    <default_value>0</default_value>
  </field>

  <field field_type="edit" advanced="true">
    <title>Server Port</title>
    <name>portforward.0.server_port</name>
    <default_value>80</default_value>
  </field>

  <field field_type="edit" advanced="true">
    <name>portforward.0.client_host</name>
    <default_value>127.0.0.2</default_value>
  </field>
</ConfigurationTemplate>
```

The example template has 5 fields: 2 normal edit fields, 2 advanced edit fields and 1 hidden field. The hidden field sets a value for `launch_type`, which cannot be changed by the user (as you would almost always do).

Custom fields

Fields of type `custom_template` are special editable fields, the values of which are expanded into other fields. In other words you can refer to the value of a custom fields in the default value of other (non-custom) fields, like this:

```
<field field_type="hidden">
  <name>command</name>
  <default_value>command -width=%(custom_template.width)</default_value>
</field>

<field field_type="custom_template">
  <title>Window Width</title>
  <name>width</name>
  <default_value>40</default_value>
</field>
```

In the example, the default value of the “command” field contains the string “%(custom_template.width)”, which refers to the value of the custom field “width” specified underneath. The value is expanded when the Menu Action is saved, so the resulting value of “command” would be “command -width=40”, assuming that the default value was used in the “width” field.

Entering default field values

Default field values are entered in the `default_value` element of a field. However some field values requires special notation:

- Boolean values: Enter 1 for True, 0 for False
- String values containing special characters : If the value contains special characters (e.g. “<” or “&”) it should be defined as CDATA in order for the XML parser not to parse it. For example, in order to add the XML string “<sender>John Smith</sender>” as a value you must enter it like this: “<default_value>![CDATA[<sender>John Smith</sender>]]</default_value>”.

Selection (drop-down)

It is possible to create a drop-down input field in the client by adding the selection element to the field. Here is an example:

```
<field field_type="custom_template">
  <name>redirectsmartcards</name>
  <title>Redirect SmartCards</title>
  <default_value>0</default_value>
  <selection>
    <selection_choice title="No" value="0" />
    <selection_choice title="Yes" value="1" />
  </selection>
</field>
```

If a field contains a selection element it will be shown as a drop-down. The selection element must contain at least one selection_choice. Each selection_choice must have a value attribute, which contains a value which can be chosen for the field. If a title attribute is specified that is what will be shown in the drop-down (like “Yes” and “No” in the example). If no title is specified the value itself will be shown in the drop-down. If default_value is set then that value is shown initially in the drop-down.

Action

An action can be added to a field, enabling the user to perform a specific action on it. In the current version only the port scan action is available:

```
<action>
  <portscan port_field="port"/>
</action>
```

The port scan action performs a scan for servers listening on a given port. The port scan action has a single required attribute port_field, which should contain the name of the field, from which the port number value should be taken.

XML schema

```

<xsd:complexType name="value_selection_choice">
  <xsd:attribute name="value" type="xsd:string" use="required" />
  <xsd:attribute name="title" type="xsd:string" use="optional" />
</xsd:complexType>

<xsd:complexType name="value_selection">
  <xsd:sequence>
    <xsd:element minOccurs="0" maxOccurs="unbounded" name="selection_choice"
type="tns:value_selection_choice"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="portscan_action">
  <xsd:attribute name="title" type="xsd:string" use="optional" default="Scan network"/>
  <xsd:attribute name="port_field" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="field_action">
  <xsd:choice maxOccurs="1" minOccurs="1">
    <xsd:element name="portscan" type="tns:portscan_action"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="field_element">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string" />
    <xsd:element minOccurs="0" name="default_value" type="xsd:string" />
    <xsd:element minOccurs="0" name="title" type="xsd:string" />
    <xsd:element minOccurs="0" name="tooltip" type="xsd:string" />
    <xsd:element minOccurs="0" name="description" type="xsd:string" />
    <xsd:element name="selection" minOccurs="0" type="tns:value_selection"/>
    <xsd:element name="action" minOccurs="0" type="tns:field_action"/>
  </xsd:sequence>
  <xsd:attribute name="field_type" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="edit" />
        <xsd:enumeration value="custom_template" />
        <xsd:enumeration value="hidden" />
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="value_type" default="string_value_type">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="string_value_type" />
        <xsd:enumeration value="text_value_type" />
        <xsd:enumeration value="integer_value_type" />
        <xsd:enumeration value="float_value_type" />
        <xsd:enumeration value="boolean_value_type" />
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="advanced" use="optional" type="xsd:boolean" default="false"/>
  <xsd:attribute name="max_length" use="optional" type="xsd:integer"/>
  <xsd:attribute name="secret" use="optional" type="xsd:boolean" default="false"/>
  <xsd:attribute name="read_only" use="optional" type="xsd:boolean" default="false"/>
  <xsd:attribute name="mandatory" use="optional" type="xsd:boolean" default="false"/>
</xsd:complexType>

<xsd:complexType name="ConfigurationTemplate">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="1" name="field" type="tns:field_element">
    </xsd:element>
    <xsd:element name="description" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required" />
  <xsd:attribute name="title" type="xsd:string" use="required" />
</xsd:complexType>

```

Field specification

In this subsection, we list all fields available in Menu Actions and a short description of their meaning. Each field has a Type, which define its value type and also defines how the input field is shown in the administration client. The types are:

String(X)	A single line string field of maximum length X.
String	A single line string field of unlimited length.
Text	A multi line string field of unlimited length.
Integer	Single line text field for integer input
Boolean	A check box

Launch type 0: Port Forward

The traditional way to enable applications for G/On is to use port forwards listening on a localhost loopback network interface. The client side application must be configured to connect to the port forward instead of directly to the server, and the G/On client will forward the connection securely to the G/On server which will forward the connection to the real server if the user is properly authorized.

Field	Type	Description
label	String(40)	The name of the Menu Action
launch_type	Integer	0 for Port Forward
command	Text	The command to launch after the port forwards has been established
working_directory	Text	If specified then this directory will be used for launching the command
close_with_process	Boolean	The port forwards will be closed when the launched command returns
kill_on_close	Boolean	The launched process will be killed when the port forward is closed
param_file_name	String	An alternative name to use for the parameter file
param_file_lifetime	Integer	The parameter file will be removed this many seconds after the command has been launched, or when the command terminates, whichever comes first.
param_file_template	Text	Create a parameter file containing this template with variables expanded
dialog_tags	String	A comma-separated list of the tags which should be attached to the Menu Action
dialog_tag_generators	Text	A newline-separated list of tag generators
portforward.0.server_host	String	The server side IP address or name to connect to
portforward.0.client_host	String	The client side IP address or name to listen on
portforward.0.server_port	Integer	The server side port number to connect to
portforward.0.client_port	Integer	The client side port number to listen on. If 0

<i>Field</i>	<i>Type</i>	<i>Description</i>
		(zero) is specified, the G/On client automatically chooses an unused port. The port that has been chosen can be observed in the variable <code>%(portforward.port)</code> - see the next section on variables.
<code>portforward.0.lock_to_process_pid</code>	Boolean	Only the launched command may use the port forward (conflicts with <code>lock_to_process_name</code>)
<code>portforward.0.sub_processes</code>	Boolean	Also allow subprocesses of the launched command to use the port forward (requires <code>lock_to_process_pid</code>)
<code>portforward.0.lock_to_process_name</code>	String	Only processes with this name are allowed to use the port forward (conflicts with <code>lock_to_process_pid</code>)

Note that additional port forwards can be added by changing the '0' in the port forward fields to '1', '2', '3', etc. The numbers must be consecutive, i.e. specifying port forwards with numbers 0 and 2 but not 1 is not allowed.

Note that many Mac OS/X applications launches and uses other processes, so features like `close_with_process`, `kill_on_close`, `lock_to_process_pid` and `sub_processes` often doesn't work as expected.

Launch type 1: Citrix Web Interface

G/On has special support for connecting to a Citrix farm through the Citrix Web Interface.

<i>Field</i>	<i>Type</i>	<i>Description</i>
<code>label</code>	String(40)	The name of the Menu Action
<code>launch_type</code>	Integer	1 for Citrix Web Interface
<code>command</code>	Text	The command to launch a browser for the Web Interface as a portforward on <code>%(portforward.host):%(portforward.port)</code> - or the end of a launch URL starting with "?"
<code>citrix_https</code>	Boolean	The connection to the Citrix Web Interface server should use HTTPS
<code>citrix_metaframe_path</code>	String	The Path part of the Web Interface URL (such as <code>"/Citrix/XenApp/"</code>)
<code>citrix_command</code>	Text	The command to launch wfica - it should reference the ICA file in <code>%(launch.param_file)</code>
<code>sso_login</code>	String	The login to use on the Web Interface - probably <code>%(user.login)</code>
<code>sso_password</code>	String	The password to use on the Web Interface - probably <code>%(user.password)</code>
<code>sso_domain</code>	String	The domain to use on the Web Interface
<code>dialog_tags</code>	String	A comma-separated list of the tags which should be attached to the Menu Action
<code>dialog_tag_generators</code>	Text	A newline-separated list of tag generators
<code>portforward.0.server_host</code>	String	Hostname or IP address of Web Interface server

<i>Field</i>	<i>Type</i>	<i>Description</i>
portforward.0.server_port	Integer	Port number on Web Interface server
portforward.0.lock_to_process_pid	Boolean	Only the launched wfica command may connect to the Citrix server (conflicts with lock_to_process_name)
portforward.0.sub_processes	Boolean	Also allow subprocesses of the launched wfica command to connect to the Citrix server (requires lock_to_process_pid)
portforward.0.lock_to_process_name	String	Only processes with this name are allowed to connect to the Citrix server (conflicts with lock_to_process_pid)
close_with_process	Boolean	The ICA connection will be closed when the launched wfica command returns

Note that Citrix client programs sometimes cooperate, so that two Citrix windows use the same wfica process and the same connection.

Launch type 3: Wake on LAN

G/On can send Wake On LAN packages to sleeping machines on the server network.

<i>Field</i>	<i>Type</i>	<i>Description</i>
label	String(40)	The name of the Menu Action
launch_type	Integer	3 for Wake on Lan
dialog_tags	String	A comma-separated list of the tags which should be attached to the Menu Action
dialog_tag_generators	Text	A newline-separated list of tag generators
portforward.0.server_host	String	UDP Target IP
portforward.0.server_port	Integer	UDP Target Port
command	Text	MAC address in the format 01:23:45:67:89:ab

Variables

A number of pre-defined values, such as loop back host address or default browser, can be added to Menu Actions fields. These variables are expanded when the Menu Action is launched. Here is a list of the variables:

Variable	Description
%(portforward.host)	The loop back host address
%(portforward.port)	The loop back port address
%(launch.param_file)	The full file name of a parameter file, if there is one
%(launch.cwd)	The current working directory, when the command is launched
%(launch.browser)	Full file name for the default browser
%(launch.ie)	Full file name for the IE browser
%(reg.<regkey>)	Value of a registry key Example of <regkey>: HKEY_CLASSES_ROOT\Applications\iexplore.exe\shell\open\command
%(env.<env.variable>)	Value of an env variable
%(cpm.ro_root)	Location where files intended for the R/O partition are placed
%(cpm.rw_root)	Location where files intended for the R/W partition are placed
%(run_env.temp)	Location of G/On temp folder
%(user.login)	User name
%(user.password)	Password in clear text
%(user.password_base64)	Password in base 64 encoding
%(user.domain) ¹	Authentication domain
%(user.netbios)16	NetBIOS name for AD Domain
%(user.my_pc_<x>)	DNS or IP address of user's PC no. x (x=1,2,3,..). Example: %(user.my_pc_1)
%(user.mac_address_<x>)	MAC address of user's PC no. x (x=1,2,3,..). Example: %(user.mac_address_1)

¹ Available only with Active Directory plugin by default. In order to get value from LDAP plugin, it must be added manually to the plugin's initialization file.