

---

SWsoft, Inc.

# Plesk Expand 2.0

## Integration Guide

Revision 2.0.2 (May 31, 2006)



(c) 2004-2006

ISBN: N/A  
SWsoft, Inc.  
13755 Sunrise Valley Drive  
Suite 325  
Herndon  
VA 20171 USA  
Phone: +1 (703) 815 5670  
Fax: +1 (703) 815 5675

Copyright © 1999-2006 by SWsoft, Inc. All rights reserved  
Distribution of this work or derivative of this work in any form is prohibited unless prior written permission is obtained from the copyright holder.

Intel, Pentium, and Celeron are registered trademarks of Intel Corporation.

MS Windows, Windows 2003 Server, Windows XP, Windows 2000, Windows NT, Windows 98, and Windows 95 are registered trademarks of Microsoft Corporation.

UNIX is a registered trademark of The Open Group.

Linux is a registered trademark of Linus Torvalds.

Fedora(tm) is a trademark of Red Hat, Inc.

RedHat is a registered trademark of Red Hat Software, Inc.

Debian(r) is a registered trademark of Software in the Public Interest, Inc.

FreeBSD is a trademark of The FreeBSD Foundation.

X Window System is a registered trademark of X Consortium, Inc.

IBM DB2 is a registered trademark of International Business Machines Corp.

SSH and Secure Shell are trademarks of SSH Communications Security, Inc.

Apache(tm) is a trademark of The Apache Software Foundation.

Xerces-C is a trademark of the Apache Software Foundation.

MySQL is a trademark of MySQL AB in the United States and other countries.

RPM is a trademark of Red Hat Software Inc.

PHP is a trademark of The PHP Group.

Perl is a trademark of O'Reilly & Associates, Inc.

JavaScript is a trademark of Sun Microsystems.

XML-RPC is a trademark of UserLand Software, Inc.

Unicode(r) is a registered trademark of Unicode, Inc.

CSS™, HTML, XHTML™ and XML are trademarks of the World Wide Web Consortium.

Curl is a trademark of Curl Corporation.

ZIP is a trademark of Iomega Corporation.

Internet Explorer is a trademark of Microsoft Corporation.

Mozilla is a trademark of the Mozilla Foundation.

Firefox is a trademark of the Mozilla Foundation.

# Contents

<b>Preface</b>	<b>4</b>
Who Should Read This Guide .....	4
About This Guide .....	4
Organization of This Guide .....	5
Documentation Conventions.....	5
Typographical Conventions.....	5
Feedback.....	6
<b>Integration Principles</b>	<b>7</b>
About Integration.....	8
How It Works .....	9
<b>Using Plesk Expand Integration API</b>	<b>11</b>
Using XML API .....	12
Composing XML Commands.....	13
Sample Statement. Creating Domain Using XML API (Locally) .....	16
Using CLI API.....	18
What is CLI in Plesk Expand .....	18
Transforming XML to CLI Input .....	19
Sample Statement. Creating Domain Using CLI API .....	20
Composing Advanced CLI Statements.....	21
Using Remote XML API .....	22
Sample Statements. Creating Domain Using Remote XML API .....	24
Using Plesk Expand Events .....	31
Configuring Plesk Expand Events.....	31
<b>Reference materials</b>	<b>35</b>
Plesk Expand Operators.....	35
Plesk Management Operators .....	35
Plesk Expand Management Operators.....	36
Obtaining Operator Usage Info .....	37
Working With Plesk Expand Operators .....	38
<b>Working with Plesk Expand Dictionary</b>	<b>39</b>
What Is Plesk Expand Dictionary? .....	39
Getting Plesk Expand Dictionary.....	40
Composing XML Query To Obtain Plesk Expand Dictionary.....	41
Obtaining Dictionary via Remote XML API.....	45
Reading Dictionary Output.....	52

---

## CHAPTER 1

# Preface

### In This Chapter

Who Should Read This Guide.....	4
About This Guide.....	4
Organization of This Guide.....	5
Documentation Conventions.....	5
Typographical Conventions .....	5
Feedback .....	6

---

## Who Should Read This Guide

This guide is intended for hosting service providers that have two or more Plesk servers to manage. Good knowledge of the operating system the provider's Plesk servers operate on is required. This can be either of the following: Linux, FreeBSD or Windows.

---

## About This Guide

The purpose of this guide is to provide comprehensive information on integration of Plesk Expand - the hosting automation software that unites multiple Plesk servers on a single hosting platform - with external systems such as billing, accounting and ordering systems. The guide fully describes the two types of integration: through the Plesk Expand Integration API (when an external system calls in Plesk Expand) and through Plesk Expand events (when Plesk Expand calls in an external system).

---

## Organization of This Guide

Chapter 2, *Integration Principles*, outlines the basic principles of Plesk Expand integration with external systems and provides some scenarios of how the integration works.

Chapter 3, *Using Plesk Expand Integration API*, focuses on the operation of the Plesk Expand Integration API.

Chapter 4, *Reference materials*, provides useful reference information about using Plesk Expand Operators.

Chapter 5, *Working with Plesk Expand Dictionary*, teaches you how to use Plesk Expand Dictionary.

---

## Documentation Conventions

Before you start using this guide, it is important to understand the documentation conventions used in it.

---

## Typographical Conventions

The following kinds of formatting in the text identify special information.

Formatting convention	Type of Information	Example
Special Bold	Items you must select, such as menu options, command buttons, or items in a list.	Go to the QoS tab.
<i>Italics</i>	Titles of chapters, sections, and subsections.  Used to emphasize the importance of a point, to introduce a term or to designate a command line placeholder, which is to be replaced with a real name or value.	Read the <b>Basic Administration</b> chapter.  The system supports the so called <i>wildcard character</i> search.
Monospace	The names of commands, files, and directories.	The license file is located in the <code>httpdocs/common/licenses</code> directory.
Preformatted	On-screen computer output in your command-line sessions; source code in XML, C++, or other programming languages.	<pre># ls -al /files total 14470</pre>

---

Preformatted Bold	What you type, contrasted with on-screen computer output.	<code># cd /root/rpms/php</code>
CAPITALS	Names of keys on the keyboard.	SHIFT, CTRL, ALT
KEY+KEY	Key combinations for which the user must press and hold down one key and then press another.	CTRL+P, ALT+F4

---

## Feedback

If you spot a typo in this guide, or if you have thought of a way to make this guide better, we would love to hear from you!

If you have a suggestion for improving the documentation (or any other relevant comments), try to be as specific as possible when formulating it. If you have found an error, please include the chapter/section/subsection name and some of the surrounding text so that we could find it easily.

Please submit a report by e-mail to [userdocs@swsoft.com](mailto:userdocs@swsoft.com).

## CHAPTER 2

# Integration Principles

## In This Chapter

About Integration .....	8
How It Works .....	9

---

## About Integration

Plesk Expand is a powerful add-on application, specially designed for managing hosting on several Plesk servers from one centralized point. Thanks to Plesk Expand hosting service providers can set up customers' domains, analyze resource usage, set rules for new client/domain creation and also integrate Plesk with external systems - such as billing systems and various in-house accounting systems.

Integration is one of the major capabilities of Plesk Expand. Plesk Expand integrates seamlessly with Plesk, which allows to provide Plesk clients and domain management services through the Plesk Expand interface, without submergence to technical details of Plesk. On the other hand, Plesk Expand integrates with a wide range of external software, such as billing, accounting, provisioning and other systems (including in-house solutions). Integration with these external systems can be achieved via Plesk Expand Integration API.

Plesk Expand Integration API includes the set of Plesk Expand operators – special programs, which have XML input and output. These Plesk Expand operators provide access to all Plesk Expand functionality. External systems call them in when communicating with Plesk Expand.

Plesk Expand Integration API provides three major integration interfaces:

- 1 *Plesk Expand XML API*. This is the main Plesk Expand API used for executing commands with specified parameters. It can be used to perform operations on the local server.
- 2 *Plesk Expand CLI (command line interface)*. You can use this API in cases when XML appears too bulky, and you wish to run simple queries directly from the command line. This API is a modification of the main XML API and is also used for executing commands on the local server.
- 3 *Plesk Expand Remote XML API* is used when it is necessary to execute the commands via XML API remotely, while maintaining the appropriate level of security.

Among the three described APIs, XML API is the main interface to be studied. Remote XML API and CLI API are based on using the XML API.

All three interfaces are based on the "call" principle. This means that you get the results by issuing queries to Plesk Expand. For example, to get Plesk Expand Dictionary you run a special query to Plesk database through Plesk Expand Integration API.

All three APIs described above are used as command line tools: the input and output appear directly in the shell.

- 4 *Plesk Expand Events*. This interface is useful if you wish to set up automatic two-way interaction between Plesk Expand and the external system.

This interface is based on the "callback" principle. Plesk Expand Events allow to monitor system activities and perform certain actions automatically when certain conditions occur in the system. For example, if a customer has created another domain, Plesk Expand notifies the system administrator about it and issues a command to the external system to bill the customer for the service.

The Plesk Expand Events API can be configured and used via Plesk Expand interface.

## How It Works

Using Plesk Expand Integration API, it is possible to integrate Plesk Expand with any third-party software, including billing and provisioning solutions, online stores, etc. This provides for building completely automated Plesk-based hosting solutions.

The scheme below illustrates the work of Plesk Expand Integration API:

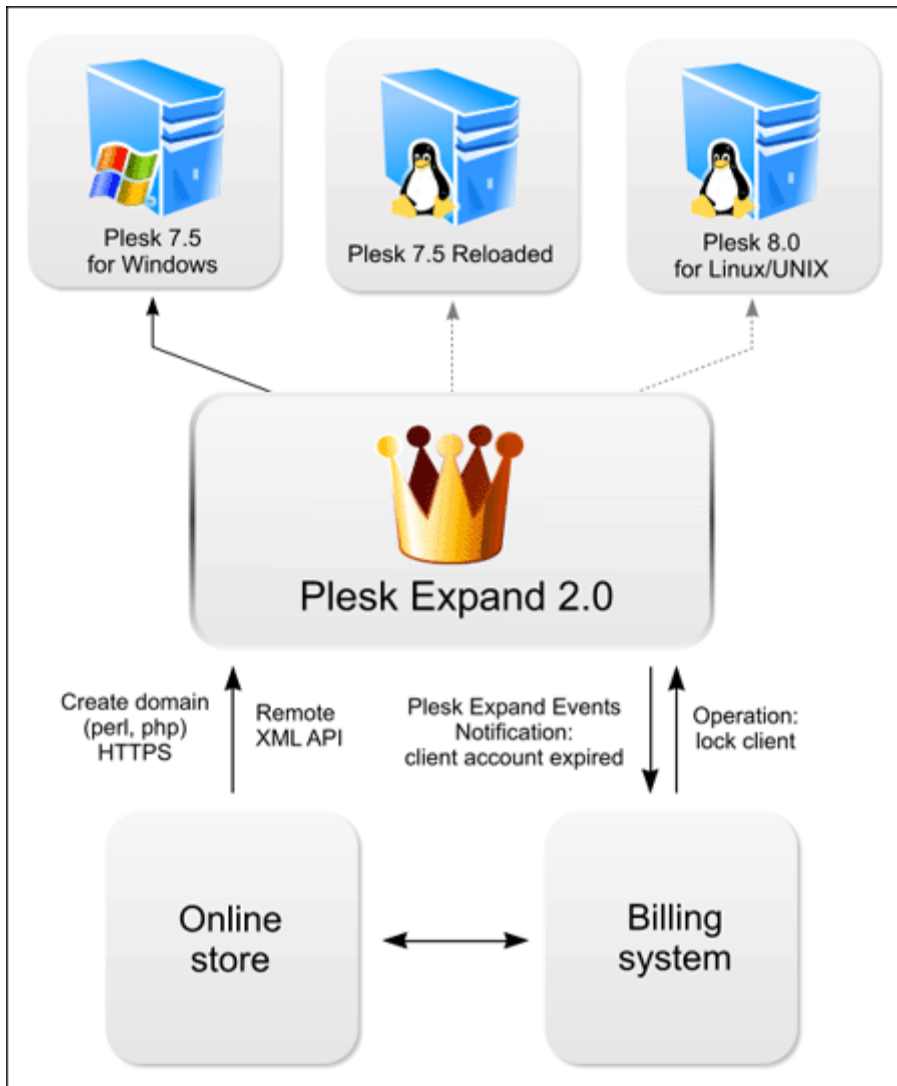


Figure 1: Plesk Expand Integration Principles

Below are typical integration scenarios.

Scenario 1:

A provider makes a decision to sell Plesk shared hosting on Linux and Windows platforms. He already has some selling/billing system and wants to use Plesk Expand for integrating this system with Plesk. No problem. He simply adds two domain templates into Plesk Expand – one template for Linux hosting and the second one for Windows hosting, makes two hosting plans “Windows Hosting” and “Linux Hosting” in his billing system, links billing plans with Plesk Expand domain template and starts selling hosting. Now new domains will be automatically created on a required platform.

Scenario 2:

At some moment a billing system sees that the client has not paid for service and provisioning, so client’s domains should be put on hold. Through the Plesk Expand Integration API the billing system calls a command tool and sets client’s domains status to disabled.

You can configure your in-house solution to perform these operations, and many more, in Plesk Expand, by calling the corresponding operators through Plesk Expand Integration API.

## CHAPTER 3

# Using Plesk Expand Integration API

This chapter focuses on the operation of the Plesk Expand Integration API.

The Plesk Expand Integration API is based on the following principles: a request for and parameters of a certain action (create/delete a domain, add/delete IP address and so on) is placed in the XML packet which is communicated to Plesk Expand together with the corresponding Plesk Expand Operator indicating the type of the action to be performed (manage a domain, manage IP addresses, etc.). All the three "call" APIs: the XML API, the Remote XML API and the CLI API, provide different ways to communicate this pair "XML query + Operator" to Plesk Expand.

Therefore, the main things you need to know is the XML API (on page 12) composing principles and the list of available Plesk Expand Operators (on page 35). Other APIs described in this manual are more or less based on this main API and are used in certain special cases:

- use CLI API when XML statement appears too bulky, and if using command line statements is more convenient for you
- use Remote XML API when you need to communicate with Plesk Expand from a remote host. This API works through the HTTPS gateway and provides secure communication with the Plesk Expand server
- use Plesk Expand Events when you need to configure automatic two-way interaction between Plesk Expand and some external application.

In this section, you will find information about the operation principles all the four APIs used in Plesk Expand: the XML API (on page 12), the CLI API (on page 18), the Remote XML API (on page 22), and Plesk Expand Events (on page 31). You will learn to compose commands for all these APIs in order to perform all available operations in Plesk Expand.

You will also find here sample statements for one of the most typical operations - creating a domain. These samples illustrate the work of three available "call" interfaces:

- the XML API - see Sample statement (on page 16)
- the CPI API - see Sample statement (on page 20)
- the Remote XML API - see Sample statements using Perl (on page 24) and PHP (on page 27) scripts.

Each sample contains detailed comments.

For details about using XML in Plesk Expand, see Composing XML Commands (on page 41).

As it was mentioned above, the Plesk Expand Integration API includes a set of Plesk Expand Integration operators, which are used for performing various tasks on Plesk Expand and Plesk hosts. A full list of available Plesk Management Operators (on page 35) and Plesk Expand Management Operators (on page 36) is provided in Chapter 4. Reference Materials.

## In This Chapter

Using XML API.....	12
Using CLI API .....	18
Using Remote XML API .....	22
Using Plesk Expand Events .....	31

---

## Using XML API

XML API is the main integration API used to perform certain actions in Plesk Expand.

You can learn how the basic operation principles of Plesk Expand XML API and learn to compose queries in subsection Composing XML Commands (on page 13), further in this section. A reader of this guide must be familiar with XML (eXtensible Markup Language) and XML Schema. More information on XML and Schema specification can be found at the World Wide Web Consortium site (<http://www.w3.org/>).

In order to perform a certain action via XML API, the following typical steps are required:

1. Compose an XML query containing all the necessary input parameters according to the guidelines set in subsection Composing XML Commands (on page 13). These XML packets are rigidly structured and this structure is described in XML Schema files (\*.xsd). You can find the Schema specification in the project documentation. The Schema usage rules are outlined in Chapter 4. Reference Materials in section Obtaining Operator Usage Info (on page 37).
2. Save this query in a separate .xml file. The filename format is arbitrary.
2. Direct the file to the standard input (stdin) of an Operator responsible for this action type. For example, if you wish to create a domain, use the `exp_plesk_domain` operator. For a full list of available Operators, refer to Plesk Expand Operators (on page 35).
3. The results are presented in the form of the XML packet which can be obtained on the standard output of the Operator.

See a sample statement for operation "Create a domain" (on page 16) further in this section.

## Composing XML Commands

A reader of this guide must be familiar with XML (eXtensible Markup Language) and XML Schema. More information on XML and Schema specification can be found at the World Wide Web Consortium site (<http://www.w3.org/>).

A usual way for any Plesk Expand operator to receive input parameters is to get an XML package on its standard input. The operator's result is also formatted as an XML packet which can be obtained on its standard output. These XML packets are rigidly structured and this structure is described in XML Schema files (\*.xsd). You can find the Schema specification in the project documentation. The Schema usage rules are outlined in **Chapter 4. Reference Materials** in section **Obtaining Operator Usage Info** (on page 37).

Hereinafter we will use standard XML idioms such as nodes, child nodes, parent nodes, sibling nodes (those which have the same parent). We will think of XML as a tree document structure. Its root is the root node of an XML packet. Descending a tree means getting child nodes, ascending means returning back to parents.

Plesk Expand uses a simplified subset of XML. The following restrictions are imposed on all Plesk Expand XML packets:

- 1 Node attributes cannot be used (except for the root node, where a single attribute '*version*' is mandatory).
- 2 A node can contain only one text child node, in this case no other child nodes can be present in that node. Such nodes are called here '*terminal*'.
- 3 Every packet has one root node called '*packet*'. Though the '*packet*' node itself doesn't limit the number of its children.

Now, let's try and open the Schema for 'exp\_plesk\_domain' input specification located in the `/usr/local/expand/share/expand/protocol/plesk/domain_input.xsd` file of Plesk Expand installation (to learn how to find the right .xsd file, see **Obtaining Operator Usage Info** (on page 37)).

If we wish to delete a domain, we need to expand the '*packet*' node, and all the children for the '*del*' node (as you can notice, the '*del*' node annotation manifests the purpose of the command, i.e. to delete something). Below you can find the description of element `<del>` in the XSD:

```
<xs:element name="del">
  <xs:annotation>
    <xs:documentation>Delete domain(s) from a database and a
related Plesk server</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="filter" type="filterType">
        <xs:annotation>
          <xs:documentation>Filter client accounts
to be deleted</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

As you can see, the *'del'* node can contain the only one node named *'filter'*. Below you can see the XSD description of element `<filter>`:

```
<xs:complexType name="filterRefrType">
  <xs:annotation>
    <xs:documentation>Filter the list of domains to be
refreshed</xs:documentation>
  </xs:annotation>
  <xs:choice>
    <xs:element name="server_id" type="id_type" minOccurs="0"
maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Plesk server
ID</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="client_id" minOccurs="0"
maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>The ID of the client the
domain owner belongs to</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="id" type="id_type" minOccurs="0"
maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Domain ID</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="plesk_domain" minOccurs="0"
maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Plesk
domain</xs:documentation>
      </xs:annotation>
      <xs:documentation>Plesk domain
ID</xs:documentation>
    </xs:element>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="server_id"
type="id_type">
          <xs:annotation>
            <xs:documentation>Server
ID</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="domain_name"
type="string">
          <xs:annotation>
            <xs:documentation>Plesk
domain name</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:choice>
</xs:complexType>
```

However, the *'filter'* node can have 3 types of children grouped by the *'choice'* type – this means that only children of the same type can be under the *'filter'* node at the same time: either *'id'*, or *'client\_id'*, or *'server\_id'*. Thus, we need a separate packet for each child mentioned. In this case, the number of the *'filter'* node children is unlimited (from zero to infinity). So, in accordance to the XML Schema we can write a packet:

```
<packet version="0.1.1.0">
  <del>
    <filter>
      <id>1</id>
      <id>3</id>
      <id>7</id>
    </filter>
  </del>
</packet>
```

This packet commands the domain operator to delete the domains with IDs 1, 3 and 7. The next packet is valid according to the XML Schema too:

```
<packet version="0.1.1.0">
  <del>
    <filter/>
  </del>
</packet>
```

This packet means we want to delete ALL of the domains Plesk Expand knows. This is a bad practice to use such packets, usually you need to specify which kind of domains you need to delete.

In addition, the schema allows deleting groups of domains:

```
<packet version="0.1.1.0">
  <del>
    <filter>
      <client_id>1</client_id>
    </filter>
  </del>
</packet>
```

Using this packet you can delete all domains owned by a client with ID 1 in Plesk Expand.

Plesk Expand operator output is specified by an XML schema too. An operator will list all the operated objects in the output packet with separate result for each object. The result at last contains the status of an object, both the error message and the code on error occurred, and also a separate ID, in case an object is unique, or a group ID.

Below are several examples of the XML output:

1. Domain was created successfully:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<packet version="2.0.1.2">
  <del>
    <result>
      <status>ok</status>
      <id>2</id>
      <client_id>0</client_id>
      <server_id>1</server_id>
    </result>
  </del>
</packet><?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

```
<packet version="2.0.1.2">
  <del>
    <result>
      <status>error</status>
      <errcode>4302</errcode>
      <errtext>Specified domain ID not found in
database</errtext>
      <id>2</id>
    </result>
  </del>
</packet>
```

## 2. Domain creation failed:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<packet version="2.0.1.2">
  <del>
    <result>
      <status>error</status>
      <errcode>4302</errcode>
      <errtext>Specified domain ID not found in
database</errtext>
      <id>2</id>
    </result>
  </del>
</packet>
```

Error: wrong domain ID.

## Sample Statement. Creating Domain Using XML API (Locally)

To create a sample domain *myowntest.domain.com* for *Client 1* on the basis of *Domain template 1*, with enabled anonymous FTP, follow these steps:

### 1. Create an XML packet describing the task, such as the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<packet version="2.0.1.2">
  <add_use_template>
    <gen_setup>
      <name>myowntest.domain.com</name>
      <client_id>1</client_id>
      <status>0</status>
      <tmpl_id>1</tmpl_id>
    </gen_setup>
    <hosting>
      <vrt_hst>
        <ftp_login>ftplot</ftp_login>
        <ftp_password>123123</ftp_password>
      </vrt_hst>
    </hosting>
  </add_use_template>
</packet>
```

In this packet, specify the following variables:

- `<name>` - the full name of the domain to be created (without the `http://www` prefix)
- `<client id>` - the ID number of the client for whom the domain is created in Plesk Expand.

- `<tmpl_id>` - the ID number of the domain template on which the domain will be based.
- `<status>0</status>` - the domain will have the "enabled" status after creation.
- `<ftp_login>` - the FTP login for this domain account.
- `<ftp_password>` - the FTP password for this domain account.

---

**Note:** For the operation to be successful, the domain template must use the *Physical hosting* hosting type.

---

2. Save this XML packet in a file - for example, `add_domain.xml`. The file name format is arbitrary.
3. Direct the file to the standard input (`stdin`) of the operator `exp_plesk_domain` used for domain creation.

The query must have the following format:

```
/usr/local/expand/sbin/exp_plesk_domain <add_domain.xml
```

---

**Note:** `exp_plesk_domain` is a Plesk Management Operator responsible for domain management operations: creating a new domain, enabling/disabling a domain, setting domain limits, configuring hosting, removing a domain, etc.

For a full list of operators, refer to the Plesk Expand Operators (on page 35) section.

---

4. The output appears in the shell, in a format similar to the following:

#### A. Domain was created successfully

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<packet version="1.2.0.27">
  <add>
    <result>
      <status>ok</status>
      <id>8</id>
    </result>
  </add>
```

- `<status>ok</status>` - operation executed successfully.
- `<id>8</id>` - the newly created domain has ID number 8.

#### B. Domain creation failed

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<packet version="2.0.1.2">
  <add_use_template>
    <result>
      <status>error</status>
      <errcode>4305</errcode>
      <errtext>[Operator] Quota problem. Domains limit was
reached for this server.</errtext>
      <client_id>11</client_id>
      <server_id>2</server_id>
    </result>
  </add_use_template>
</packet>
```

Error: domains limit was reached for this server.

---

## Using CLI API

### What is CLI in Plesk Expand

Plesk Expand operators are not intended to be run manually. Instead there's a frontend interface to the Plesk Expand Task Manager which calls them. But sometimes it's necessary to use an operator directly from the command line. In this case a problem arises: XML is too bulky to be used in its full-scale as a command line input argument.

Plesk Expand operators have an additional ability to accept command line input arguments, which they then convert to XML (edible for the operator itself). We assume this interface is more comprehensive and actually reduces typing by 50%-70% rate.

We will refer to the command line interface as CLI in this guide.

Further in this section you will find guidelines on composing CLI statements in subsection Transforming XML to CLI Input (on page 19), and a sample CLI statement (on page 20) for one of the most typical operations - creating a domain.

## Transforming XML to CLI Input

Let's first consider a simple example of XML input to the `exp_plesk_ev` operator:

```
<packet version="0.1.1.0">
  <del>
    <filter>
      <id>3</id>
    </filter>
  </del>
</packet>
```

Here's how it must look like in the CLI variant:

```
$ exp_plesk_domain del filter id=3
```

As it can be seen a default behavior for the CLI input parser is to get the next argument and put it inside the previous one (the '*packet*' node is not needed here, it will be inserted automatically). If an argument contains the '=' sign a node gets assigned a text data value, which comes directly after the '=' sign.

Here is one more example of XML input for `exp_expand_config` operator:

```
<packet version="0.1.1.0">
  <person>
    <add>
      <login>nobody</login>
      <email>nobody@test.com</email>
      <name>Nobody person</name>
      <password>setup</password>
      <roles>
        <role_id>1</role_id>
      </roles>
    </add>
  </person>
</packet>
```

That is how it looks like in CLI input:

```
$ exp_expand_config person add login=nobody email=nobody@localhost \
  name='Nobody person' password=setup roles role_id=1
```

As you can see, if a node gets its text data, it's considered a terminal one (i.e. having no children). Thus we can just enlist the sibling terminal nodes. In case we want to put '*roles*' node as the first child of the '*add*' node, the CLI input will look like this:

```
$ exp_expand_config person add roles [ role_id=1 ] \
  email=nobody@localhost name='Nobody person' \
  password=some-pass
```

Note the square brackets. They enclose all the children for a preceding node (in our example, the node '*role\_id*'). And the XML packet in this case will look as follows:

```
<packet version="0.1.1.0">
  <person>
    <add>
      <roles>
        <role_id>1</role_id>
      </roles>
      <email>nobody@test.com</email>
      <name>Nobody person</name>
      <password>setup</password>
```

```

        </add>
    </person>
</packet>

```

After a square bracket gets closed, the siblings for that node follow. You may say that this package is rather simple and there is no difference whether to use square brackets or not. But when you read this section further and consider more complicated examples, you will see how useful the square brackets are.

---

**Note:** Each bracket must be separated with at least one space from the arguments on both sides. This restriction stems from the method the shell uses to build the command line arguments list for a program. It just drops quotes.

---

Thus one can't tell these two structures one from another:

```

$ exp_expand_config person add name='Some weird [person]'
$ exp_expand_config person add name='Some weird '[person]'

```

Both of them look identical from the shell's point of view. By the way, that's why an error is generated for such "obvious" shell commands:

```

$ [-n "foo" ] && echo OK

```

---

**Note:** Please remember: no spaces before or after the equation sign ('=').

---

## Sample Statement. Creating Domain Using CLI API

Before running this script, make sure that the following conditions are met:

- at least one Plesk server is registered in Plesk Expand
- at least one client (Plesk client or Expand client) is registered in Plesk Expand
- at least one domain template is created in Plesk Expand.

For details about performing the above listed operation, refer to *Getting Started with Plesk Expand* here (<http://www.swsoft.com/en/products/plesk/expand/>).

To create a domain using CLI API, perform the following steps:

1. Compose and run a CLI statement according to the guidelines provided in the example below and in subsection Transforming XML to CLI Input (on page 19), further in this section.

The sample provides a CLI input you should send to Plesk Expand in order to create a domain *myowntest.domain.com* for *Client 1* on the basis of *Domain template 1*, with enabled anonymous FTP:

```

/usr/local/expand/sbin/exp_plesk_domain add_use_template gen_setup [
name=myowntest.domain.com client_id=1 status=0 tmp1_id=1 ] hosting [
vrt_hst ftp_login=ftlog ftp_password=123123 ]

```

This statement uses the same variables as the XML API sample (on page 16) provided in section Using XML API (on page 12).

2. The output is provided in XML; it has the same format and provides the same information as the output of a regular XML API statement. See the XML API Sample statement. Creating domain using XML API (Locally) (on page 16) for details.

## Composing Advanced CLI Statements

Below are some more advanced examples of using CLI for operators:

This packet is valid for `exp_plesk_domain` operator:

```
<packet version="0.1.1.0">
  <refresh>
    <filter>
      <server_id>3</server_id>
    </filter>
    <dataset>
      <gen_info/>
      <stat/>
      <prefs/>
      <limits/>
      <hosting/>
      <user/>
    </dataset>
  </refresh>
</packet>
```

This packet tells Plesk Expand to reload data (general information, statistics, preferences, limits, physical hosting and user information) of Plesk server with ID=3.

CLI variant:

```
$ exp_plesk_domain refresh filter [ server_id=3 ] dataset gen_info- \
  stat- prefs- limits- hosting- user-
```

To reload the same set of data on several Plesk servers, provide these servers IDs in square brackets:

```
$ exp_plesk_domain refresh filter [ server_id=1 server_id=2
server_id=3 ] dataset gen_info- \
  stat- prefs- limits- hosting- user-
```

For terminal nodes, which have neither text data values nor other child nodes, it's good to use a '-' sign at the end of their names. This construction is also valid (though lengthier):

```
$ exp_plesk_domain refresh filter [ server_id=1 server_id=2
server_id=3 ] dataset \
  gen_info [ ] stat [ ] prefs [ ] limits [ ] hosting [ ] \
  user [ ]
```

One more example:

```
<packet version="0.1.1.0">
  <add>
    <gen_setup>
      <name>Basic</name>
      <htype>vrt_hst</htype>
    </gen_setup>
    <hosting>
      <vrt_hst>
        <fp>>false</fp>
        <fp_ssl>>false</fp_ssl>
        <fp_auth>>false</fp_auth>
        <use_shell>>true</use_shell>
      </vrt_hst>
    </hosting>
```

```
</add>
</packet>
```

This packet tells Plesk Expand to create a domain template *Basic*, with hosting type *physical hosting*, with permission *shell access* enabled and other specified permissions *disabled*. Direct this packet to the standard input of operator `exp_plesk_tmpl_domain`.

CLI variant of this packet:

```
$ exp_plesk_tmpl_domain add gen_setup [ name='First of all' \
    htype=vrt_hst ] hosting vrt_hst [ fp=false fp_ssl=false \
    fp_auth=false use_shell=true ]
```

If this seems bogus for reading, try to put more brackets, e.g. in the following way:

```
$ exp_plesk_tmpl_domain add [ \
    gen_setup [ name='First of all' htype=vrt_hst ] \
    hosting [ vrt_hst [ fp=false fp_ssl=false fp_auth=false \
    use_shell=true ] ] ]
```

---

## Using Remote XML API

*Plesk Expand Remote XML API* is used when you need to perform certain operations from a remote host.

In Remote XML API, the XML packet is wrapped in the PHP or Perl script which contains all the required information: administrator's access details, the name of the corresponding Plesk Expand Operator and other relevant data.

A reader of this section must be familiar with PHP and Perl programming languages. For more information on these technologies, refer to the [www.php.net](http://www.php.net) and [www.perl.com](http://www.perl.com) websites.

In order to perform a certain action via Remote XML API, the following steps are required:

1. Compose a script in Perl or PHP for connecting to the remote Plesk Expand host over HTTPS and transferring the necessary data. The URL for connecting to this host through the webgate will have the following format: <https://your.domain.com:8442/webgate.php>

The script must contain the following information:

- the server you should perform operations on (the Plesk Expand host)
- administrator login
- administrator password
- the Plesk Expand Operator to be used

---

**Note:** For a full list of available operators, refer to Plesk Expand Operators (on page 35) section.

The full list of necessary commands and parameters with detailed comments are provided in the descriptions of Perl (on page 24) and PHP (on page 27) samples.

2. Insert a standard XML API packet containing the parameters of the query into the body of the script. For guidelines on creating XML API statements, see Composing XML Commands (on page 13).

3. Save the statement in a separate `.pl` or `.php` file; the filename format is arbitrary. Execute the file.

4. The results are provided in the form of the XML packet.

See a sample statement for operation "Create a domain" using Perl (on page 24) or PHP (on page 27) scripts further in this section.

## Sample Statements. Creating Domain Using Remote XML API

In this section, you will examples of using Perl and PHP scripts for executing command "create a domain" via Remote XML API. Before running this script, make sure that the following conditions are met:

- at least one Plesk server is registered in Plesk Expand
- at least one client (Plesk client or Expand client) is registered in Plesk Expand
- at least one domain template is created in Plesk Expand.

For details about performing the above listed operation, refer to *Getting Started with Plesk Expand* here (<http://www.swsoft.com/en/products/plesk/expand/>).

### Sample in Perl

To create a sample domain *myowntest.domain.com* for *Client 1* on the basis of *Domain template 1*, with enabled anonymous FTP, follow these steps:

```
#!/usr/bin/perl
use strict;
use Data::Dumper;
use Net::SSLeay qw(get_https make_form make_headers post_https);
$Net::SSLeay::ssl_version = 3;
my $host = "server1.com";
my $port = 8442;
my $login = 'root';
my $script = "/webgate.php";
my $passwd = 'setup';
my $operator = 'exp_plesk_domain';
my $request = <<EOL;

<?xml version="1.0" encoding="UTF-8"?>
<packet version="2.0.1.2">
  <add_use_template>
    <gen_setup>
      <name>myowntest.domain.com</name>
      <client_id>1</client_id>
      <status>0</status>
      <tmpl_id>1</tmpl_id>
    </gen_setup>
    <hosting>
      <vrt_hst>
        <ftp_login>ftplot</ftp_login>
        <ftp_password>123123</ftp_password>
      </vrt_hst>
    </hosting>
  </add_use_template>
</packet>
EOL

my $headers = make_headers (
  'HTTP_AUTH_LOGIN' => $login
  , 'HTTP_AUTH_PASSWD' => $passwd
  , 'HTTP_AUTH_OP' => $operator
  , 'Content-Type' => 'text/xml'
);
print "----- REQUEST -----<n";
```

```

print "REQUEST_HEADERS=".Dumper($headers)."\n";
print "REQUEST DATA\n".$request."\n";
print "----- RESULT -----\n";
my ($reply_data, $reply_type, %reply_headers) =
post_https($host, $port, $script, $headers, $request);
print "REPLY_HEADERS=".Dumper(\%reply_headers)."\n";
print "REPLY_TYPE=".$reply_type."\n";
print "REPLY_DATA:\n".$reply_data."";

```

For detailed explanations on the syntax and parameters of this statement, see **Variables and Operations and Comments** further in this subsection.

### Variables

In this statement, there are the following variables:

Variable	Description
my \$host	Provide the name or IP address of the Plesk host on which you wish to create a domain.
my \$port	The standard port used for communication with Plesk Expand. The default meaning is 8442; if you have set another port for communication, specify it here.
my \$login	Provide your Plesk Expand login.
my \$passwd	Provide your Plesk Expand password.
my \$operator	Provide the name of the Plesk Management Operator responsible for domain creation. For the full list of operators, refer to section Plesk Management Operators (on page 35).  For domain management operations, <code>exp_plesk_domain</code> is used.

### Operations and comments

In this statement, the following functions/groups of functions perform the following tasks:

Operation / group of operations	Action performed
<pre> use Data::Dumper;  use Net::SSLeay qw(get_https make_form make_headers post_https);  \$Net::SSLeay::ssl_version = 3; </pre>	Calling up the necessary libraries.

<pre>my \$request = &lt;&lt;EOL; &lt;...&gt; EOL</pre>	<p>The XML packet providing the parameters of the domain to be created.</p> <p>Variables in the XML packet: &lt;name&gt;, &lt;client_id&gt;, &lt;tmpl_id&gt; and others are explained in section Sample Statement. Creating Domain Using XML API (on page 16).</p>
<pre>my \$script = "/webgate.php";</pre>	<p>A Plesk Expand script used for remote calls.</p>
<pre>my \$headers = make_headers ( 'HTTP_AUTH_LOGIN'=&gt; \$login ,'HTTP_AUTH_PASSWD' =&gt; \$passwd ,'HTTP_AUTH_OP'=&gt; \$operator ,'Content-Type'=&gt;'text/xml' );</pre>	<p>Creating the packet header before sending it to Plesk Expand.</p>
<pre>print "----- REQUEST ----- \n";  print "REQUEST_HEADERS=".Dumper(\$headers)." \n";  print "REQUEST DATA\n".\$request." \n";  print "----- RESULT ----- \n";</pre>	<p>Sending request to stdout.</p>
<pre>my (\$reply_data, \$reply_type, %reply_headers) = post_https(\$host, \$port, \$script, \$headers, \$request);</pre>	<p>Sending the packet to the remote server.</p>
<pre>print "REPLY_HEADERS=".Dumper(\%reply_headers)." \n" ;  print "REPLY_TYPE=".\$reply_type." \n";  print "REPLY_DATA:\n".\$reply_data."";</pre>	<p>Setting the parameters of the server reply to stdout:</p> <p>REPLY_HEADERS - the reply header</p> <p>REPLY_TYPE - the reply type</p> <p>REPLY_DATA - the reply data.</p>

## Sample in PHP

To create a sample domain *myowntest.domain.com* for *Client 1* on the basis of *Domain template 1*, with enabled anonymous FTP, follow these steps:

```
<?php
function write_callback($ch, $data) {
    echo $data;
    return strlen($data);
}

function sendCommand($operator, $data, $login, $passwd, $host,
    $port=8442) {
    $script = "webgate.php";
    $url = "https://$host:$port/$script";
    $headers = array(
        "HTTP_AUTH_OP: $operator",
        "HTTP_AUTH_LOGIN: $login",
        "HTTP_AUTH_PASSWD: $passwd",
        "Content-Type: text/xml",
    );

    $ch = curl_init();
    curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, FALSE);
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, FALSE);
    curl_setopt($ch, CURLOPT_HTTPHEADER, &$amp;headers);
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_WRITEFUNCTION, write_callback);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
    curl_setopt($ch, CURLOPT_VERBOSE, 1);
    $result = curl_exec($ch);
    if (!$result) {
        echo "\n\n-----\nncURL error
number:".curl_errno($ch);
        echo "\nncURL error:".curl_error($ch);
    }

    curl_close($ch);
    return;
}

$data = <<<EOF
<?xml version="1.0" encoding="UTF-8"?>
<packet version="2.0.1.2">
    <add_use_template>
        <gen_setup>
            <name>myowntest.domain.com</name>
            <client_id>1</client_id>
            <status>0</status>
            <tmpl_id>1</tmpl_id>
        </gen_setup>
        <hosting>
            <vrt_hst>
                <ftp_login>ftplot</ftp_login>
                <ftp_password>123123</ftp_password>
            </vrt_hst>
        </hosting>
    </add_use_template>
</packet>
```

```

EOF;

$operator="exp_plesk_domain";
$login="root";
$password="setup";
$host="server1.com";
$port=8442;

sendCommand($operator, $data, $login, $pass, $host, $port);
?>

```

For detailed explanations on the syntax and parameters of this statement, see **Variables** and **Operations and Comments** further in this subsection.

### Variables

In this statement, replace the following variables with the appropriate content:

Variable	Description
\$host	Provide the name or IP address of the Plesk host on which you wish to create a domain.
\$port	The standard port used for communication with Plesk Expand. The default meaning is 8442; if you have set another port for communication, specify it here.
\$login	Provide your Plesk Expand login.
\$password	Provide your Plesk Expand password.
\$operator	Place the XML packet containing the operation parameters. Provide the name of the Plesk Management Operator responsible for domain creation. For the full list of operators, refer to section Plesk Management Operators (on page 35).  For domain management operations, <code>exp_plesk_domain</code> is used.

### Operations and comments

In this statement, the following functions/groups of functions perform the following tasks:

Operation / group of operations	Action performed
<pre> function write_callback(\$ch, \$data) { echo \$data; return strlen(\$data); } </pre>	Declaring function: getting server reply to stdout.
<pre> function sendCommand(\$operator, \$data, \$login, \$password, \$host, \$port=8442) </pre>	Declaring function: generate and send the packet to Plesk Expand.

<pre>\$script = "webgate.php";  \$url = "https://\$host:\$port/\$script";</pre>	<p>A Plesk Expand script used for remote calls. The URL is composed from the described variables and commands:</p> <p>\$host, \$port and \$script.</p>
<pre>\$headers = array(  "HTTP_AUTH_OP: \$operator",  "HTTP_AUTH_LOGIN: \$login",  "HTTP_AUTH_PASSWD: \$passwd",  "Content-Type: text/xml",  );</pre>	<p>Creating packet header.</p>
<pre>\$ch = curl_init();  curl_setopt(\$ch, CURLOPT_SSL_VERIFYHOST, FALSE);  curl_setopt(\$ch, CURLOPT_SSL_VERIFYPEER, FALSE);  curl_setopt(\$ch, CURLOPT_HTTPHEADER, &amp;\$headers);  curl_setopt(\$ch, CURLOPT_URL, \$url);  curl_setopt(\$ch, CURLOPT_WRITEFUNCTION, write_callback);  curl_setopt(\$ch, CURLOPT_POSTFIELDS, \$data);  curl_setopt(\$ch, CURLOPT_VERBOSE, 1);</pre>	<p>Initializing the curl engine: setting HTTPS parameters, packet header (\$headers), specifying the URL to be processed (\$url), callback functions, etc.</p>
<pre>\$result = curl_exec(\$ch);</pre>	<p>Sending data to the remote server and requesting output.</p>
<pre>\$data = &lt;&lt;&lt;EOF  &lt;...&gt;  EOF;</pre>	<p>The XML packet providing the parameters of the domain to be created.</p> <p>Variables in the XML packet: &lt;name&gt;, &lt;client_id&gt;, &lt;tmpl_id&gt; and others are explained in section Sample Statement. Creating Domain Using XML API (on page 16).</p>

```
sendCommand($operator, $data, $login,  
$pass, $host, $port);
```

Calling the function to create  
and send the packet to the  
remote server.

---

# Using Plesk Expand Events

## Configuring Plesk Expand Events

This chapter focuses on the callback communication of Plesk Expand with external systems, i.e. when Plesk Expand calls in an external system. This communication happens thanks to Plesk Expand events, which can be configured so that they could transmit a definite set of parameters to the necessary external system.

Events in Plesk Expand are divided into classes. At the present moment, there are eleven event classes in Plesk Expand.

- **Element was unlocked forcibly** - occurs when a certain operation that is being performed with a certain object in the system gets suddenly aborted; the system locks this object, then after a certain period of time unlocks it.
- **Person password reminder** - occurs when a person forgets the password and uses a password reminder option available from the control panel login screen; the system sends the password to this user e-mail.
- **Plesk server network status changed** - occurs when, for example, a certain server gets disconnected due to the network problem and then gets connected to the network again.
- **Plesk server resource quota exceeded** - occurs when, for example, you try to create a new client or a domain on a server, but the limits on the maximum number of clients or domains that can be registered on this server exceeded.
- **Plesk client operated** - occurs when any changes are made to a client (creation, update, removal).
- **Plesk DNS zone operated** - occurs when an action is performed on a DNS zone (a DNS zone is created, updated or removed).
- **Plesk domain operated** - occurs when a domain is operated on (created, updated or removed).
- **Plesk license operated** - occurs when an action is performed on a Plesk license key (it is updated, removed or a new key is added).
- **Plesk package operated** - occurs when a site application package is operated on (updated, removed or a new package is created).
- **Plesk server operated** - occurs when a certain action is performed on a Plesk server (it is updated, removed or a new server is registered in the system).
- **Too many actions failed per hour** - occurs when the number of failed actions exceeds the value of 'max\_failed\_user\_actions' / 'max\_failed\_auto\_actions' parameters set in the expand.conf. file.

For each event class you may set two types of event handlers:




- Mail notification;
- Run Programs.

This chapter will concentrate on the 'Run Program' event handler, which is capable of running a set of programs for any Plesk Expand event.

You may configure 'Run Program' event handler in two ways: through web-based interface (see Plesk Expand User Guide (<http://www.swsoft.com/en/products/plesk/expand20/docs/>) for details) or through the Plesk Expand Integration API.

## Setting 'Run Program' Handler Through Plesk Expand Events

To set a 'run program' handler through web-based interface, do the following:

- 1 Select the  Server shortcut in the navigation pane. Click the  Events Setup icon on the Server administration page.
- 2 Select the desired event class from the list of event classes by clicking its name. On the Event Class management page, click Run programs.
- 3 Click the  New Program icon on the Run programs page to add a new program for the 'run program' event handler.
- 4 On the New Program page, fill in the following fields:

Field Name	Description	Example
Command Line	The command for program execution with possible arguments in the command line (use placeholders, given under the Environmental Data field, if you like)	<code>/usr/bin/my_handler -s &lt;@server_name&gt;</code>
Standard Input Data	Data transmitted to the program standard input (use placeholders, given under the Environmental Data field, if you like)	<code>&lt;@server_name&gt;</code>
Environment Data	The list of environmental variables available for the executed program (use placeholders, given under the Environmental Data field, if you like)	<code>server=&lt;@server_name&gt; status=&lt;@status&gt;</code>

5 Click **OK** to submit the settings and to return to the **Run programs** page.

This program will be added to the list of programs and will be called in each time the event this program is set for occurs in the system. To update this program, select it by clicking its name and edit data in the required fields.

When Run Program event handler is being executed it scans for programs registered for execution on a triggered event. A registered program has:

```
the command line specification suitable for feeding to 'sh -c', for
example: echo 'Some event occurred' > /tmp/exp_events.log
```

- the command standard input;
- the command environment (the only environment a command will get).

Then the programs are run in an arbitrary order. By default programs `stdout` is open to `/dev/null`. To log it somewhere else just add shell redirection, e.g. `> /var/log/some.log`. Programs `stderr` will be collected by Event Manager and if it was run from Task Manager it will appear in the Action Log. To suppress errors just use shell redirection again. If a program dies of an unhandled signal or returns exit code different from 0 this is interpreted as a program failed execution. All programs are run without respect to other programs execution statuses.

## Setting 'Run Program' Handler Through API

You may also set a 'Run Program' event handler through Plesk Expand Integration API.

'Run Program' event handler is implemented as an operator (`exp_expand_ev_runprog`). For it to handle events some programs for execution must be added. This can be done with the Run Program Event Handler Configuration operator (`exp_expand_evconf_runprog`). For more information on its capabilities look at its XSD specification (located in the `share/expand/protocol/expand/ev_handlers/evconf_runprog_input.xsd` file of the Plesk Expand installation).

Here's a simple example of input for the **Person password reminder** event class:

```
<packet version="1.0">
  <add>
    <filter>
      <ev_class>expand_person_password_remind</ev_class>
    </filter>
    <values>
      <cmd_line>echo 'Some event occurred' >
/tmp/exp_events.log</cmd_line>
      <cmd_env>
        FOO=BAR
        BOO=ZOO
      </cmd_env>
      <cmd_spec>This will go to stdin</cmd_spec>
    </values>
  </add>
</packet>
```

After a command line for an event is added it gets a unique identifier. Using it one can delete a command line or tune it. Here is an example of a command for deletion:

```
<del>
  <filter>
```

```
        <id>10</id>  
        <id>11</id>  
        <id>12</id>  
    </filter>  
</del>
```

## CHAPTER 4

# Reference materials

## In This Chapter

Plesk Expand Operators .....	35
------------------------------	----

---

## Plesk Expand Operators

At this moment Plesk Expand provides two groups of operators - Plesk Management Operators and Plesk Expand Management Operators.

### Plesk Management Operators

The following operators are used for performing various operations with Plesk:

Operator	Full Name	Explanation
exp_plesk_server	Plesk Server Management	Allows performing operations with Plesk servers, such as registering a new server, editing administrator's info, changing password, managing IP-pool, removing a server, etc.
exp_plesk_client	Plesk Client Management	Used for creating a new Plesk client account, editing client info, activating/deactivating a client account, setting permissions for operations and limits on resource usage, removing a client account, etc.
exp_plesk_multi_client	Expand Client Management	Used for creating a new Expand client account, editing client info, activating/deactivating a client account, setting permissions for operations and limits on resource usage, removing a client account, etc.
exp_plesk_domain	Plesk Domain Management	Serves for creating a new domain, enabling/disabling a domain, setting domain limits, configuring hosting, removing a domain, etc.
exp_plesk_siteapp	Management of client's Application pools	Used for performing operations with the Application pool: requesting information about the packages contained in the pool, adding and removing applications to/from pool, etc.

exp_plesk_ip	Plesk Server IP Addresses Management	Used for creating IP ranges, assigning IP addresses to particular domains, searching for free IP addresses, removing IP addresses, etc.
exp_plesk_dns	Plesk Server DNS Records Management	Allows managing DNS zones, adding new DNS records, removing them, etc.
exp_plesk_tmpl_domain	Plesk Domain Template Management	Serves for creating domain templates, editing and removing them.
exp_plesk_tmpl_client	Plesk Client Template Management	Allows creating client templates, editing template info and removing a client template.
exp_plesk_dictionary	Plesk Dictionary Obtaining	Allows to get a Plesk dictionary (an information snapshot of available limits, permissions and physical hosting services) of any Plesk version for any supported operating system.

## Plesk Expand Management Operators

The following operators are used for performing various operations with Plesk Expand:

Operator	Full Name	Explanation
exp_expand_config	Plesk Expand Configuration	Used for changing Plesk Expand configuration and for managing persons, action log manipulation, etc.
exp_expand_evconf	Plesk Expand Event Configuration	Serves for configuring event classes and their handlers.
exp_expand_evconf_notif	Plesk Expand Notifications Event Handler Configuration	Allows configuring mail-notification handlers.
exp_expand_ev_runprog	Plesk Expand 'Run Program' Event Handler Configuration	Used for configuring 'run program' event handlers.

## Obtaining Operator Usage Info

Currently Plesk Expand operators do not support generating usage (i.e., help) on input. Instead the original XML schemas must be used (\* .xsd files describing which input is valid for an operator, i.e. the allowed XML tags and their parent-children relations). They are better seen with XMLSpy for Windows:

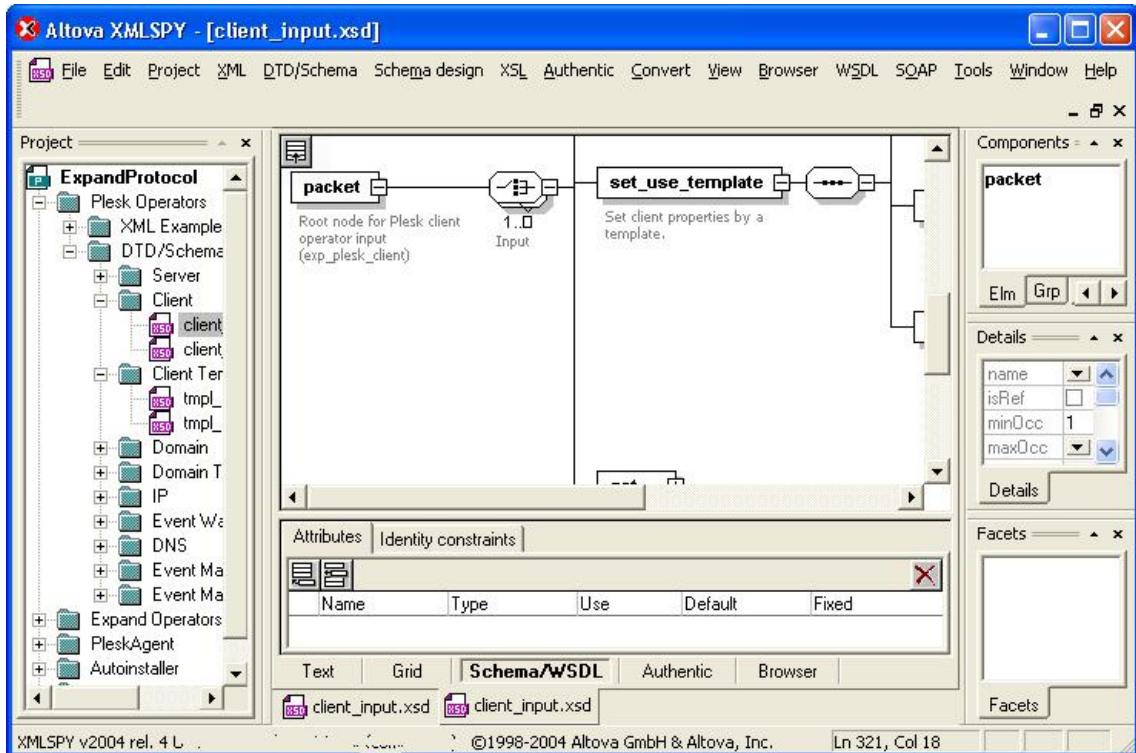


Figure 2: XML Spy Project

Also XML schemas are available in HTML format.

Plesk Expand is shipped with the `expand-api-*.tgz` documentation tarball. XML Spy project file is available in the tarball at the path `protocol/common/protocol/ExpandProtocol.spp`. Documentation in HTML format is available in the “html” folder.

You can use the following table for easy navigation in API documentation files:

Operator	XMLSpy project files	HTML documentation files
exp_plesk_server	PleskOperators/Schemas/Server/*.xsd	html/expand/plesk/server_input.html html/expand/plesk/server_output.html
exp_plesk_client	PleskOperators/Schemas/Client/*.xsd	html/expand/plesk/client_input.html html/expand/plesk/client_output.html
exp_plesk_domain	PleskOperators/Schemas/Domain/*.xsd	html/expand/plesk/domain_input.html html/expand/plesk/domain_output.html

exp_plesk_ip	PleskOperators/Schemas/IP/*.xsd	html/expand/plesk/ip_input.html html/expand/plesk/ip_output.html
exp_plesk_dns	PleskOperators/Schemas/DNS /*.xsd	html/expand/plesk/dns_input.html html/expand/plesk/dns_output.html
exp_plesk_tmpl_domain	PleskOperators/Schemas/Domain Template/*.xsd	html/expand/plesk/tmpl_domain_input.html html/expand/plesk/tmpl_domain_output.html
exp_plesk_tmpl_client	PleskOperators/Schemas/Client Templates/*.xsd	html/expand/plesk/tmpl_client_input.html html/expand/plesk/tmpl_client_output.html

## Working With Plesk Expand Operators

Plesk Expand operators have an XML API and can be called in the following ways:

- through the Command Line Interface (locally or remotely);
- through the HTTPS gateway (remotely).

It should be noted that a web-based Plesk Expand frontend at low level also calls in Plesk Expand operators. The detailed info on these call-ins can be found in the **Action Log**. (For more information on the **Action Log**, please refer to **Plesk Expand User's Guide**.)

## CHAPTER 5

# Working with Plesk Expand Dictionary

This chapter describes the process of obtaining and using Plesk Expand Dictionary.

## In This Chapter

What Is Plesk Expand Dictionary? .....	39
Getting Plesk Expand Dictionary .....	40

---

## What Is Plesk Expand Dictionary?

*Plesk Expand Dictionary* is a brief information summary which provides information about limits, permissions and physical hosting services available in all supported Plesk versions, and their default meanings. You can obtain a full dictionary of all supported Plesk versions, or filter the dictionary by Plesk version and operating system.

The dictionary is available even in case no Plesk servers have been installed or registered in Plesk Expand yet. This feature allows you to know which services are available in each Plesk version prior to purchasing and installing this version on your system.

It is possible to view dictionary in any language pack installed with Plesk Expand 2.0. By default, English language pack is used (en\_US).

To obtain a dictionary, direct the XML packet containing the parameters of the query to the standard input of the corresponding Plesk Management Operator `exp_plesk_dictionary`. For details about composing XML queries (on page 41) and Remote XML API (on page 22) statements for obtaining dictionary, see section Getting Plesk Expand Dictionary (on page 40).

---

## Getting Plesk Expand Dictionary

To get the Plesk dictionary on all supported Plesk versions, or on a particular Plesk version, you need to run a query to the Plesk Expand database via XML API or Remote XML API. For details about composing the queries, see sections [Using XMI API](#) (on page 12) and [Using Remote XML API](#) (on page 22) in [Chapter 2. Using Plesk Expand Integration API](#).

For details about creating XML setting filtering parameters for the Dictionary, see [Composing XML Query To Obtain Plesk Expand Dictionary](#) (on page 41).

For examples of perl or PHP statements used for obtaining the Dictionary via Remote XML API, see [Obtaining Dictionary via Remote XML API](#) (on page 45).

The output is provided in the form of XML packet. See [Reading Dictionary](#) (on page 52) to know how to read the dictionary output.

## Composing XML Query To Obtain Plesk Expand Dictionary

Inside the script, place an XML packet (one of provided below or a similar one) to obtain particular information about all Plesk versions, one or several particular versions of interest, or version(s) under a particular operation system.

You can use the following types of packets:

- To get a full list of Plesk versions supported by Expand:

Syntax:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<packet version="1.1.0.4">
  <list/>
</packet>
```

In this packet there are no variables; you can copy and paste it as is.

- To get a list of Plesk versions running Unix (or Windows, accordingly):

Syntax:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<packet version="1.1.0.4">
  <list>
    <version_os>OS</version_os>
  </list/>
</packet>
```

Example:

In this sample, we request full information about Plesk for 8.0 for Linux/UNIX, including a list of available limits (client and domain-level), permissions, and physical hosting services, and have asked to present the output in English.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<packet version="1.1.0.4">
  <list>
    <version_os>unix</version_os>
  </list/>
</packet>
```

OR

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<packet version="1.1.0.4">
  <list>
    <version_os>windows</version_os>
  </list/>
</packet>
```

In this packet, specify the following variable:

<version\_os> - the operation system of interest: *windows* or *unix*.

- If you wish to know information on several versions, use the sample presented above and the <filter> tag. Specify the version numbers and operating systems for all Plesk versions of interest

Syntax:

```
<filter>
  <version>
    <plesk_version>x.x</plesk_version>
    <plesk_os>OS</plesk_os>
  </version>
  <version>
    <plesk_version>x.x</plesk_version>
    <plesk_os>OS</plesk_os>
  </version>
</filter>
```

Example:

In this packet, we request full information about Plesk 8.0 for Linux/UNIX and Plesk 7.5 for Windows.

```
<filter>
  <version>
    <plesk_version>8.0</plesk_version>
    <plesk_os>unix</plesk_os>
  </version>
  <version>
    <plesk_version>7.5</plesk_version>
    <plesk_os>windows</plesk_os>
  </version>
</filter>
```

In this packet, specify the following variables:

<plesk\_version> - the number of the Plesk version of interest.

<version\_os> - the operation system of interest: *windows* or *unix*.

- To get full information about the capabilities provided by a particular Plesk version, in a particular language

Syntax:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<packet version="1.1.0.5">
  <get>
    <filter>
      <version>
        <plesk_version>x.x</plesk_version>
        <plesk_os>OS</plesk_os>
      </version>
    </filter>
    <dataset>
      <limits>
        <client />
        <domain />
      </limits>
      <perms>
        <client />
      </perms>
      <hosting>
        <vrt_hst />
      </hosting>
    </dataset>
    <locale>xx_XX</locale>
  </get>
```

```
</packet>
```

Example:

In this packet, we request information about limits, permissions and physical hosting services provided by Plesk 8.0 for Linux/Unix, and the default values of these parameters.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<packet version="1.1.0.5">
  <get>
    <filter>
      <version>
        <plesk_version>8.0</plesk_version>
        <plesk_os>unix</plesk_os>
      </version>
    </filter>
    <dataset>
      <limits>
        <client />
        <domain />
      </limits>
      <perms>
        <client />
      </perms>
      <hosting>
        <vrt_hst />
      </hosting>
    </dataset>
    <locale>en_US</locale>
  </get>
</packet>
```

In this packet, specify the following variables:

<plesk\_version> - the number of the Plesk version of interest.

<version\_os> - the operation system of interest: *windows* or *unix*

<locale> - the language you wish to receive the output in. The *en\_US* input corresponds to English (US), which is the default Plesk Expand language pack.

---

**Note.** If the corresponding language pack is unavailable, the output will be displayed in the language of the *base locale*. *Base locale* is a default locale which had been specified during the Expand installation process.

---

In this statement, the following functions/groups of functions perform the following tasks:

Operation/group of operations	Action performed
-------------------------------	------------------

<pre>&lt;dataset&gt;   &lt;limits&gt;     &lt;client /&gt;     &lt;domain /&gt;   &lt;/limits&gt;   &lt;perms&gt;     &lt;client /&gt;   &lt;/perms&gt;   &lt;hosting&gt;     &lt;vrt_hst /&gt;   &lt;/hosting&gt; &lt;/dataset&gt;</pre>	<p>Requesting information about the Dictionary available in this version: limits, permissions and physical hosting services.</p> <p>You can filter the output by excluding the corresponding nodes from the packet.</p> <p>If you do not need information, for example, about the permissions available in the desired Plesk version, exclude the &lt;perms&gt; node and all its contents:</p> <pre>&lt;perms&gt;   &lt;client /&gt; &lt;/perms&gt;.</pre>
<pre>&lt;limits&gt;   &lt;client /&gt;   &lt;domain /&gt; &lt;/limits&gt;</pre>	<p>Requesting information about limits (inside the general statement).</p> <p>There are client and domain-level limits in Plesk Expand. If you do not need information, for example, about domain limits, exclude the &lt;domain /&gt; node from the packet.</p>
<pre>&lt;perms&gt;   &lt;client /&gt; &lt;/perms&gt;</pre>	<p>Requesting information about permissions (inside the general statement). Permissions are not subdivided into client and domain-level permissions.</p>
<pre>&lt;hosting&gt;   &lt;vrt_hst /&gt; &lt;/hosting&gt;</pre>	<p>Requesting information about physical hosting services (inside the general statement).</p>

## Obtaining Dictionary via Remote XML API

To obtain a dictionary via Remote XML API, run a Perl or PHP script and wrap a corresponding XML query in the body of the script.

Before running this script, make sure that the following conditions are met:

- at least one Plesk server is registered in Plesk Expand
- at least one client (Plesk client or Expand client) is registered in Plesk Expand
- at least one domain template is created in Plesk Expand.

For details about performing the above listed operation, refer to *Getting Started with Plesk Expand* here (<http://www.swsoft.com/en/products/plesk/expand/>).

To obtain a dictionary of one or several Plesk versions, follow these steps:

1. Create an XML packet according to the guidelines set in *Composing XML Query To Obtain Plesk Dictionary* (on page 41). In the XML query, specify the parameters of the operation: the filtering parameters (full dictionary, dictionaries for one/several Plesk versions, filtered by operating system), and set the language of the output (English, German, Spanish, Russian or Japanese).

2. Compose a Perl or PHP script according to the guidelines provided in *Using Remote XML API* (on page 22) and save it in the `.pl` or `.php` file, correspondingly. The filename format is arbitrary.

You can see the sample Perl (on page 46) and PHP (on page 49) statements used for obtaining Dictionary further in this section.

3. Run the file.

## Sample in Perl

To obtain a dictionary of *Plesk 8.0 for Linux/UNIX* in the language of the *base locale*, run the following script:

```
#!/usr/bin/perl
use strict;
use Data::Dumper;

use Net::SSLeay qw(get_https make_form make_headers post_https);
$Net::SSLeay::ssl_version = 3;
my $host = "server1.com";
my $port = 8442;
my $script = "/webgate.php";
my $login = 'root';
my $passwd = 'setup';
my $operator = 'exp_plesk_dictionary'
my $request = <<EOL;

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<packet version="1.1.0.5">
  <get>
    <filter>
      <version>
        <plesk_version>8.0</plesk_version>
        <plesk_os>unix</plesk_os>
      </version>
    </filter>
    <dataset>
      <limits>
        <client />
        <domain />
      </limits>
      <perms>
        <client />
      </perms>
      <hosting>
        <vrt_hst />
      </hosting>
    </dataset>
  </get>
</packet>
EOL

my $headers = make_headers (
  'HTTP_AUTH_LOGIN'=> $login
  , 'HTTP_AUTH_PASSWD' => $passwd
  , 'HTTP_AUTH_OP'=> $operator
  , 'Content-Type'=>'text/xml'
);
print "----- REQUEST -----<n";
print "REQUEST_HEADERS=" .Dumper($headers)."<n";
print "REQUEST DATA<n". $request."<n";
print "----- RESULT -----<n";
my ($reply_data, $reply_type, %reply_headers) =
post_https($host, $port, $script, $headers, $request);
print "REPLY_HEADERS=" .Dumper(\%reply_headers)."<n";
print "REPLY_TYPE=" . $reply_type."<n";
print "REPLY_DATA:<n". $reply_data."<n";
```

For detailed explanations on the syntax and parameters of this statement, see **Variables** and **Operations and Comments** further in this subsection.

### Variables

In this statement, there are the following variables:

Variable	Description
<code>my \$host</code>	Provide the name or IP address of the Plesk host on which you wish to create a domain.
<code>my \$port</code>	The standard port used for communication with Plesk Expand. The default meaning is 8442; if you have set another port for communication, specify it here.
<code>my \$login</code>	Provide your Plesk Expand login.
<code>my \$passwd</code>	Provide your Plesk Expand password.
<code>my \$operator</code>	Provide the name of the Plesk Management Operator responsible for obtaining Plesk dictionary. For the full list of operators, refer to section Plesk Management Operators (on page 35).  For obtaining Plesk Expand dictionary, <code>exp_plesk_dictionary</code> is used.

### Operations and comments

In this statement, the following functions/groups of functions perform the following tasks:

Operation / group of operations	Action performed
<pre>use Data::Dumper;  use Net::SSLeay qw(get_https make_form make_headers post_https);  \$Net::SSLeay::ssl_version = 3;</pre>	Calling up the necessary libraries.
<pre>my \$request = &lt;&lt;EOL;  &lt;...&gt;  EOL</pre>	<p>The XML packet providing the Dictionary filtering parameters.</p> <p>Variables in the XML packet: <code>&lt;plesk_version&gt;</code>, <code>&lt;plesk_os&gt;</code> and others, are explained in section Composing XML Query To Obtain Plesk Expand Dictionary (on page 41).</p>

<pre>my \$script = "/webgate.php";</pre>	<p>A Plesk Expand script used for remote calls.</p>
<pre>my \$headers = make_headers ( 'HTTP_AUTH_LOGIN'=&gt; \$login , 'HTTP_AUTH_PASSWD' =&gt; \$passwd , 'HTTP_AUTH_OP'=&gt; \$operator , 'Content-Type'=&gt;'text/xml' );</pre>	<p>Creating the packet header before sending it to Plesk Expand.</p>
<pre>print "----- REQUEST ----- \n";  print "REQUEST_HEADERS=".Dumper(\$headers)."\n";  print "REQUEST DATA\n".\$request."\n";  print "----- RESULT ----- \n";</pre>	<p>Sending request to stdout.</p>
<pre>my (\$reply_data, \$reply_type, %reply_headers) = post_https(\$host, \$port, \$script, \$headers, \$request);</pre>	<p>Sending the packet to the remote server.</p>
<pre>print "REPLY_HEADERS=".Dumper(\%reply_headers)."\n" ;  print "REPLY_TYPE=".\$reply_type."\n";  print "REPLY_DATA:\n".\$reply_data."";</pre>	<p>Setting the parameters of the server reply to stdout:</p> <p>REPLY_HEADERS - the reply header</p> <p>REPLY_TYPE - the reply type</p> <p>REPLY_DATA - the reply data.</p>

## Sample in PHP

To obtain a dictionary of *Plesk 8.0 for Linux/UNIX* in the language of the *base locale*, run the following script:

```
<?php
function write_callback($ch, $data) {
    echo $data;
    return strlen($data);
}

function sendCommand($operator, $data, $login, $passwd, $host,
    $port=8442) {
    $script = "webgate.php";
    $url = "https://$host:$port/$script";

    $headers = array(
        "HTTP_AUTH_OP: $operator",
        "HTTP_AUTH_LOGIN: $login",
        "HTTP_AUTH_PASSWD: $passwd",
        "Content-Type: text/xml",
    );

    $ch = curl_init();
    curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, FALSE);
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, FALSE);
    curl_setopt($ch, CURLOPT_HTTPHEADER, &$amp;headers);
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_WRITEFUNCTION, write_callback);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
    curl_setopt($ch, CURLOPT_VERBOSE, 1);
    $result = curl_exec($ch);

    if (!$result) {
        echo "\n\n-----\ncURL error
number:".curl_errno($ch);
        echo "\n\ncURL error:".curl_error($ch);
    }
    curl_close($ch);
    return;
}

$data = <<<EOF
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<packet version="1.1.0.5">
    <get>
        <filter>
            <version>
                <plesk_version>8.0</plesk_version>
                <plesk_os>unix</plesk_os>
            </version>
        </filter>
        <dataset>
            <limits>
                <client />
                <domain />
            </limits>
            <perms>
                <client />
            </perms>
        </dataset>
    </get>
</packet>
</EOF>
```

```

        </perms>
        <hosting>
            <vrt_hst />
        </hosting>
    </dataset>
</get>
</packet>
EOF;

$operator="exp_plesk_dictionary";
$login="root";
$password="setup";
$host="server1.com";
$port=8442;

sendCommand($operator, $data, $login, $pass, $host, $port);
?>

```

For detailed explanations on the syntax and parameters of this statement, see **Variables** and **Operations and Comments** further in this subsection.

### Variables

In this statement, replace the following variables with the appropriate content:

Variable	Description
\$host	Provide the name or IP address of the Plesk host on which you wish to create a domain.
\$port	The standard port used for communication with Plesk Expand. The default meaning is 8442; if you have set another port for communication, specify it here.
\$login	Provide your Plesk Expand login.
\$password	Provide your Plesk Expand password.
\$operator	Provide the name of the Plesk Management Operator responsible for obtaining Plesk dictionaries. For the full list of operators, refer to section Plesk Management Operators (on page 35).  For obtaining Plesk Expand dictionary, <code>exp_plesk_dictionary</code> is used.

### Operations and comments

In this statement, the following functions/groups of functions perform the following tasks:

Operation / group of operations	Action performed
<pre> function write_callback(\$ch, \$data) { echo \$data; return strlen(\$data); </pre>	Declaring function: getting server reply to stdout.

<pre>function sendCommand()</pre>	<p>Declaring function: generate and send the packet to Plesk Expand.</p>
<pre>\$headers = array( "HTTP_AUTH_OP: \$operator", "HTTP_AUTH_LOGIN: \$login", "HTTP_AUTH_PASSWD: \$pass", "Content-Type: text/xml", );</pre>	<p>Creating packet header.</p>
<pre>\$data = &lt;&lt;&lt;EOF &lt;...&gt; EOF;</pre>	<p>The XML packet providing the Dictionary filtering parameters.</p> <p>Variables in the XML packet: &lt;plesk_version&gt;, &lt;plesk_os&gt; and others are explained in section Composing XML Query To Obtain Plesk Expand Dictionary (on page 41).</p>
<pre>\$url = "https://\$host:\$port/\$script" ;</pre>	<p>Composing of the appropriate URL from the specified variables:</p> <p>\$host, \$port and \$script.</p>
<pre>\$script = "webgate.php" ;</pre>	<p>A Plesk Expand script used for remote calls.</p>
<pre>\$ch = curl_init(); curl_setopt(\$ch, CURLOPT_SSL_VERIFYHOST, FALSE); curl_setopt(\$ch, CURLOPT_SSL_VERIFYPEER, FALSE); curl_setopt(\$ch, CURLOPT_HTTPHEADER, &amp;\$headers); curl_setopt(\$ch, CURLOPT_URL, \$url); curl_setopt(\$ch, CURLOPT_WRITEFUNCTION, write_callback); curl_setopt(\$ch, CURLOPT_POSTFIELDS, \$data); curl_setopt(\$ch, CURLOPT_VERBOSE, 1);</pre>	<p>Initializing the curl engine: setting HTTPS parameters, packet header (\$headers), specifying the URL to be processed (\$url), callback functions, etc.</p>
<pre>\$result = curl_exec(\$ch);</pre>	<p>Sending data to the remote server and requesting output.</p>

```
sendCommand($operator, $data, $login,
$pass, $host, $port);
```

Calling the function to create and send the packet to the remote server.

## Reading Dictionary Output

The output appears directly in the shell in XML format. The representation of limits, permissions and physical hosting services in the dictionary is explained further in this section.

Plesk dictionaries use a special terminology in which the information is provided. Below there are several examples illustrating typical dictionary outputs containing information about limits, permissions and physical hosting services available in a certain Plesk version, with their default values.

- Client limit "Disk space":

```
<limits>
  <client>
    <limit>
      <limit_name>max_mg</limit_name>
      <localized_value>Maximum number of mail
groups.</localized_value>
      <value_min>0</value_min>
      <value_max>999999</value_max>
      <value_unl>-1</value_unl>
      <units>
        <unit_name>items</unit_name>
        <localized_value>items</localized_value>
        <default>>true</default>
      </units>
    </limit> ..
```

This entry provides the following information:

Value	Meaning
<code>&lt;limit_name&gt;max_mg&lt;/limit_name&gt;</code>	The name of the limit in Plesk database.
<code>&lt;localized_value&gt;Maximum number of mail groups.&lt;/localized_value&gt;</code>	The name of the limit in Plesk and Plesk Expand control panel.
<code>&lt;value_min&gt;0&lt;/value_min&gt;</code> <code>&lt;value_max&gt;99999999&lt;/value_max&gt;</code>	This limit is specified as a number from 0 to 99999999.
<code>&lt;value_unl&gt;-1&lt;/value_unl&gt;</code>	You can set this limit to "unlimited" by entering <i>-1</i> .
<code>&lt;unit_name&gt;items&lt;/unit_name&gt;</code>	This limit is set in unit of measurement "items".
<code>&lt;default&gt;&gt;true&lt;/default&gt;</code>	In this Plesk version this feature is enabled by default. If this line reads <i>false</i> , the feature is disabled.

- Domain limit "Validity period":

```
<limits>
  <domain>
    <limit>
      <limit_name>expiration</limit_name>
      <localized_value>Validity period.</localized_value>
      <value_min>0</value_min>
      <value_max>2115914400</value_max>
      <value_unl>-1</value_unl>
      <units>
        <unit_name>hours</unit_name>
        <localized_value>hours</localized_value>
        <default>>false</default>
        <unit_name>days</unit_name>
        <localized_value>days</localized_value>
        <default>>true</default>
        <unit_name>weeks</unit_name>
        <localized_value>weeks</localized_value>
        <default>>false</default>
        <unit_name>months</unit_name>
        <localized_value>Months</localized_value>
        <default>>false</default>
        <unit_name>years</unit_name>
        <localized_value>years</localized_value>
        <default>>false</default>
      </units>
    </limit> ...
  </domain>
</limits>
```

This entry provides the following information:

Value	Meaning
<code>&lt;limit_name&gt;expiration&lt;/limit_name&gt;</code>	The name of the limit in Plesk database.
<code>&lt;localized_value&gt;Validity period.&lt;/localized_value&gt;</code>	The name of the limit in Plesk and Plesk Expand control panels.
<code>&lt;value_min&gt;0&lt;/value_min&gt;</code>	This limit is specified as a number between 0 and 2115914400.
<code>&lt;value_max&gt;2115914400&lt;/value_max&gt;</code>	
<code>&lt;value_unl&gt;-1&lt;/value_unl&gt;</code>	You can set the limit to "unlimited" by entering -1.
<code>&lt;unit_name&gt;days&lt;/unit_name&gt;</code>	This limit is set in unit of measurement "days".
<code>&lt;localized_value&gt;days&lt;/localized_value&gt;</code>	
<code>&lt;default&gt;&gt;true&lt;/default&gt;</code>	

<pre> &lt;unit_name&gt;hours&lt;/unit_name&gt;  &lt;localized_value&gt;hours&lt;/localized_value&gt;      &lt;default&gt;false&lt;/default&gt; ... &lt;unit_name&gt;weeks&lt;/unit_name&gt;  &lt;localized_value&gt;weeks&lt;/localized_value&gt;      &lt;default&gt;false&lt;/default&gt; ... &lt;unit_name&gt;months&lt;/unit_name&gt;  &lt;localized_value&gt;Months&lt;/localized_value&gt;      &lt;default&gt;false&lt;/default&gt; ... &lt;unit_name&gt;years&lt;/unit_name&gt;  &lt;localized_value&gt;years&lt;/localized_value&gt;      &lt;default&gt;false&lt;/default&gt; </pre>	<p>By default, the limit is set in <i>days</i>, but you may also set it in <i>hours</i>, <i>weeks</i>, <i>months</i> or <i>years</i>.</p>
--	---

- Permission "Physical hosting management".

```

<perms>
  <client>
    <perm>
      <perm_name>manage_hosting</perm_name>
      <localized_value>Physical hosting
management.</localized_value>
      <value_dflt>>false</value_dflt>
    </perm>
    <perm> ...
  </perms>

```

This output provides the following information:

Value	Meaning
<code>&lt;perm_name&gt;manage_phosting&lt;/perm_name&gt;</code>	The name of the limit in Plesk database.
<code>&lt;localized_value&gt;Physical hosting management.&lt;/localized_value&gt;</code>	The name of the limit in Plesk and Plesk Expand control panels.
<code>&lt;value_dflt&gt;&gt;false&lt;/value_dflt&gt;</code>	By default, this permission is <i>disabled</i> .

- Physical hosting service "ASP" (ability for clients to deploy ASP applications on their websites):

```
<hosting>
  <vrt_hst>
    <phosting>
      <service>asp</service>
      <value_dflt>>false</value_dflt>
    </phosting>
  </vrt_hst>
</hosting>
```

This output provides the following information:

Value	Meaning
<code>&lt;service&gt;asp&lt;/service&gt;</code>	The name of the physical hosting service.
<code>&lt;value_dflt&gt;&gt;false&lt;/value_dflt&gt;</code>	For clients using this Plesk version, using ASP is <i>disallowed</i> by default.