



## Cisco Access Registrar Tcl and REX Dictionaries

---

This appendix describes the Tcl and REX dictionaries that are used when writing Incoming or Outgoing scripts.

A dictionary is a data structure that contains key/value pairs. Two types of dictionaries exist: the Attribute dictionaries (used by the Request and Response dictionaries), and the Environment dictionary.

This section contains the dictionaries you reference when writing a Tcl script and the dictionaries you reference when you write a script using the shared libraries (REX—RADIUS EXtension).

### Tcl Attribute Dictionaries

An *Attribute dictionary* is a dictionary in which the keys are constrained to be the names of attributes as defined in the Cisco Access Registrar server configuration, and the values are the string representation of the legal values for that particular attribute. For example, IP addresses are specified by the dotted-decimal string representation of the address, and enumerated values are specified by the name of the enumeration. This means numbers are specified by the string representation of the number.

Attribute dictionaries have the unusual feature that there can be more than one instance of a particular key in the dictionary. These instances are ordered, with the first instance at index zero. Some of the methods of an Attribute dictionary allow an index to be specified to indicate a particular instance or position in the list of instances to be referenced.

### Attribute Dictionary Methods

Attribute dictionaries use active commands, called *methods*, that allow you to change and access the values in the dictionaries. [Table A-1](#) lists of all of the methods you can use with the Request and Response dictionaries.

Table A-1 Tcl Attribute Dictionary Methods

Name	Syntax	Description
<b>addProfile</b>	<b>\$dict addProfile</b> <profile> [<mode>]	Copies all of the attributes in the profile <profile> into the dictionary. Note, <profile> must be the name of one of the profiles listed in the server configuration. When <mode> is not provided or when <mode> equals the special value <b>REPLACE</b> , any duplicate instances of the attributes in the dictionary are replaced with the attribute from <profile>. When <mode> is provided and equals the special value <b>APPEND</b> , new instances of the attributes are appended to the attributes already in the dictionary. When <mode> is provided and equals the special value <b>AUGMENT</b> , only add the attribute when it does not already exist.
<b>clear</b>	<b>\$dict clear</b>	Removes all entries from the dictionary.
<b>containsKey</b>	<b>\$dict containsKey</b> <attribute>	Returns 1 when the dictionary contains the attribute <attribute>, otherwise returns 0.
<b>firstKey</b>	<b>\$dict firstKey</b>	Returns the name of the first attribute in the dictionary. Note, the attributes are not stored sorted by name.
<b>get</b>	<b>\$dict get</b> <attribute> [<index> [bMore]]	Returns the value of the <attribute> attribute from the dictionary, represented as a string. When the dictionary does not contain the <attribute>, an empty string is returned.  When <index> is provided, return the <index>'th instance of the attribute. Some attributes can appear more than once in the request (or response) packet. The <index> argument is used to select which instance to return.  When <b>bMore</b> is provided, the <b>get</b> method sets <b>bMore</b> to 1 when more attributes exist after the one returned, and to 0 otherwise. You can use this to determine whether another call to <b>get</b> should be made to retrieve other instances of the attribute.
<b>isEmpty</b>	<b>\$dict isEmpty</b>	Returns 1 when the dictionary has no entries, otherwise returns 0.
<b>log</b>	<b>\$dict log</b> <level> <message> ...	Outputs a message into the RADIUS server's logging system. The <level> should be either <b>LOG_ERROR</b> , <b>LOG_WARNING</b> , or <b>LOG_INFO</b> . The remaining arguments are concatenated together and sent to the logging system at the specified level.
<b>nextKey</b>	<b>\$dict nextKey</b>	Returns the name of the next attribute in the dictionary that follows the attribute returned in the last call to <b>firstKey</b> or <b>nextKey</b> .
<b>put</b>	<b>\$dict put</b> <attribute> <value> [<index>]	Associates <value> with the attribute <attribute> in the dictionary. When <index> is not provided or when <index> equals the special value <b>REPLACE</b> , any existing instances of <attribute> are replaced with the single value. When <index> is provided and equals the special value <b>APPEND</b> , a new instance of <attribute> is appended to the end of the list of instances of the <attribute>. When <index> is provided and is a number, a new instance of <attribute> is inserted at the position indicated. When <index> is provided and equals the special value <b>AUGMENT</b> , only put the attribute when it does not already exist.

Table A-1 Tcl Attribute Dictionary Methods (continued)

Name	Syntax	Description
remove	<code>\$dict remove &lt;attribute&gt; [&lt;index&gt;]</code>	Removes the <attribute> attribute from the dictionary. When <index> is not provided or when <index> equals the special value <b>REMOVE_ALL</b> , remove any existing instances of <attribute>. When <index> is provided and is a number, remove the instance of <attribute> at the position indicated.  Always returns 1, even when the dictionary did not contain the <attribute> at that <index>.
size	<code>\$dict size</code>	Returns the number of entries in the dictionary.
trace	<code>\$dict trace &lt;level&gt; &lt;message&gt; ...</code>	Outputs a message into the packet tracing system used by the RADIUS server. At level 0, no tracing occurs. At level 1, only an indication the server received the packet and sent a reply is output. As the number gets higher, the amount of information output increases, until at level 4, where everything is traced as output. The remaining arguments are concatenated and sent to the tracing system at the specified level.

## Tcl Environment Dictionary

A dictionary is a data structure that contains key/value pairs. An Environment dictionary is a dictionary in which the keys and values are constrained to be strings. The Tcl Environment dictionary is used to communicate information from the script to the server and from script to script within the processing of a particular request. Note, there can be only one instance of a key in the Environment dictionary.

Table A-2 lists of all the methods you can use with the Request and Response dictionaries.

Table A-2 Tcl Environment Dictionary Methods

Name	Syntax	Description
clear	<code>\$dict clear</code>	Removes all entries from the dictionary.
containsKey	<code>\$dict containsKey &lt;key&gt;</code>	Returns 1 when the dictionary contains the <key> key, otherwise returns 0.
firstKey	<code>\$dict firstKey</code>	Returns the name of the first key in the dictionary. Note, the keys are not stored sorted by name.
get	<code>\$dict get &lt;key&gt;</code>	Returns the value of <key> from the dictionary. When the dictionary does not contain the <key>, an empty string is returned.
isEmpty	<code>\$dict isEmpty</code>	Returns 1 when the dictionary has no entries, otherwise returns 0.
log	<code>\$dict log &lt;level&gt; &lt;message&gt; ...</code>	Outputs a message into the logging system used by the RADIUS server. <level> should be one of <b>LOG_ERROR</b> , <b>LOG_WARNING</b> , or <b>LOG_INFO</b> . The remaining arguments are concatenated together and sent to the logging system at the specified level.
nextKey	<code>\$dict nextKey</code>	Returns the name of the next key in the dictionary that follows the key returned in the last call to <b>firstKey</b> or <b>nextKey</b> .
put	<code>\$dict put &lt;key&gt; &lt;value&gt;</code>	Associates <value> with the <key> key in the dictionary, replacing an existing instance of <key> with the new value.
remove	<code>\$dict remove &lt;key&gt;</code>	Removes the <key> key from the dictionary. Always returns 1, even when the dictionary did not contain the <key>.

Table A-2 Tcl Environment Dictionary Methods (continued)

Name	Syntax	Description
size	<code>\$dict size</code>	Returns the number of entries in the dictionary.
trace	<code>\$dict trace &lt;level&gt; &lt;message&gt;</code> ...	Outputs a message into the packet tracing system used by the RADIUS server. At level 0, no tracing occurs. At level 1, only an indication the server received the packet and sent a reply is output. As the number gets higher, the amount of information output is greater, until at level 4, where everything the server traces is output. The remaining arguments are concatenated together and sent to the tracing system at the specified level.

## REX Attribute Dictionary

A dictionary is a data structure that contains key/value pairs. An Attribute dictionary is a dictionary in which the keys are constrained to be the attributes as defined in the RADIUS server configuration and the values are constrained to be legal values for that particular attribute. Attribute dictionaries have the unusual feature that there can be more than one instance of a particular key in the dictionary. These instances are ordered, with the first instance at index 0. Some of the methods of an Attribute dictionary allow an index to be specified to indicate a particular instance or position in the list of instances to be referenced.

When writing REX scripts, you can specify keys as the string representation of the name of the attribute or by type, which is a byte sequence defining the attribute. The values can also be specified as the string representation of the value or as the byte sequence, which is the attribute. These options mean some of these access methods have four different variations that are the combinations of string or type for the key, and string or bytes for the value.

## Attribute Dictionary Methods

Attribute dictionaries use active commands, called *methods*, that allow you to change and access the values in the dictionaries.

[Table A-3](#) lists all of the methods you can use with the Request and Response dictionaries.

Table A-3 REX Attribute Dictionary Methods

Name	Syntax	Description
<b>addProfile</b>	<code>abool_t pDict-&gt;addProfile(rex_AttributeDictionary_t* pDict, const char* &lt;pszProfile&gt;, int &lt;iMode&gt;)</code>	Copies all of the attributes in the <pszProfile> profile into the dictionary. Note, <pszProfile> must be the name of one of the profiles listed in the server configuration. When <iMode> equals the special value <b>REX_REPLACE</b> , it replaces any duplicate instances of the attributes in the dictionary with the attribute from the profile. When <iMode> equals the special value <b>REX_APPEND</b> , it appends a new instance of the attributes to any attributes already in the dictionary. When <iMode> equals the special value <b>REX_AUGMENT</b> , it only puts the attribute when does not already exist.
<b>allocateMemory</b>	<code>void* pDict-&gt;allocateMemory(rex_AttributeDictionary_t* pDict, unsigned int &lt;iSize&gt;)</code>	Allocates memory for use in scripts that persist only for the lifetime of this request. This memory is released when processing for this request is complete.
<b>clear</b>	<code>void pDict-&gt;clear(rex_AttributeDictionary_t* pDict)</code>	Removes all entries from the dictionary.
<b>containsKey</b>	<code>abool_t pDict-&gt;containsKey(rex_AttributeDictionary_t* pDict, const char* &lt;pszAttribute&gt;)</code>	Returns TRUE when the dictionary contains <pszAttribute>, otherwise returns FALSE.
<b>containsKeyByType</b>	<code>abool_t pDict-&gt;containsKeyByType(rex_AttributeDictionary_t* pDict, const abytes_t* &lt;pAttribute&gt;)</code>	Returns TRUE when the dictionary contains <pAttribute>, otherwise returns FALSE.
<b>firstKey</b>	<code>const char* pDict-&gt;firstKey(rex_AttributeDictionary_t* pDict)</code>	Returns the name of the first attribute in the dictionary. Note, the attributes are not stored sorted by name.
<b>firstKeyByType</b>	<code>const abytes_t* pDict-&gt;firstKeyByType(rex_AttributeDictionary_t* pDict)</code>	Returns a pointer to the byte sequence defining the first attribute in the dictionary. Note, attributes are not stored sorted by name.
<b>get</b>	<code>const char* pDict-&gt;get(rex_AttributeDictionary_t* pDict, const char* pszAttribute, int &lt;iIndex&gt;, abool_t* &lt;pbMore&gt;)</code>	Returns the value of the <iIndex>'d instance of the attribute from the dictionary, represented as a string. When the dictionary does not contain the attribute (or that many instances of the attribute), an empty string is returned.  When <pbMore> is non-zero, the <b>get</b> method sets <pbMore> to TRUE when more instances of the attribute exist after the one returned, and to FALSE otherwise. This can be used to determine whether another call to <b>get</b> should be made to retrieve other instances of the attribute.

Table A-3 REX Attribute Dictionary Methods (continued)

Name	Syntax	Description
<b>getBytes</b>	<code>const abytes_t* pDict-&gt;getBytes(rex_AttributeDictionary_t* pDict, const char* pszAttribute, int &lt;iIndex&gt;, abool_t* &lt;pbMore&gt;)</code>	<p>Returns the value of the &lt;iIndex&gt;'d instance of the attribute from the dictionary, as a sequence of bytes. When the dictionary does not contain the attribute (or that many instances of the attribute), 0 is returned.</p> <p>When &lt;pbMore&gt; is non-zero, the <b>getBytes</b> method sets &lt;pbMore&gt; to TRUE when more instances of the attribute exist after the one returned, and to FALSE otherwise. This can be used to determine whether another call to <b>getBytes</b> should be made to retrieve other instances of the attribute.</p>
<b>getBytesByType</b>	<code>const abytes_t* pDict-&gt;getBytesByType(rex_AttributeDictionary_t* pDict, const abytes_t* pAttribute, int &lt;iIndex&gt;, abool_t* &lt;pbMore&gt;)</code>	<p>Returns the value of the &lt;iIndex&gt;'d instance of the attribute from the dictionary, as a sequence of bytes. When the dictionary does not contain the attribute (or that many instances of the attribute), 0 is returned instead.</p> <p>When &lt;pbMore&gt; is non-zero, sets the variable pointed to TRUE when more instances of the attribute exist after the one returned, and to FALSE otherwise. This can be used to determine whether another call to <b>get</b> should be made to retrieve other instances of the attribute.</p>
<b>getByType</b>	<code>const char* pDict-&gt;get(rex_AttributeDictionary_t* pDict, const abytes_t* &lt;pszAttribute&gt;, int &lt;iIndex&gt;, abool_t* &lt;pbMore&gt;)</code>	<p>Returns the value of the &lt;iIndex&gt;'d instance of the attribute from the dictionary, as represented as a string. When the dictionary does not contain the attribute (or that many instances of the attribute), returns an empty string.</p> <p>When &lt;pbMore&gt; is non-zero, the <b>getByType</b> method sets &lt;pbMore&gt; to TRUE when more instances of the attribute exist after the one returned, and to FALSE otherwise. This can be used to determine whether another call to <b>getByType</b> should be made to retrieve other instances of the attribute.</p>
<b>getType</b>	<code>const char* pDict-&gt;getByType(rex_AttributeDictionary_t* pDict, const abytes_t* &lt;pAttribute&gt;)</code>	Returns a pointer to the byte sequence defining the attribute, when the attribute name matches a configured attribute, zero otherwise.
<b>isEmpty</b>	<code>abool_t pDict-&gt;isEmpty(rex_AttributeDictionary_t* pDict)</code>	Returns TRUE when the dictionary has 0 entries, FALSE otherwise.

Table A-3 REX Attribute Dictionary Methods (continued)

Name	Syntax	Description
log	<code>abool_t pDict-&gt;log(rex_AttributeDictionary_t* pDict, int &lt;iLevel&gt;, const char* &lt;pszFormat&gt;, ...)</code>	Outputs a message into the logging system used by the RADIUS server. <iLevel> should be one of <b>REX_LOG_ERROR</b> , <b>REX_LOG_WARNING</b> , or <b>REX_LOG_INFO</b> . The <b>pszFormat</b> argument is treated as a <b>printf</b> -style format string, and it, along with the remaining arguments, are formatted and sent to the logging system at the specified level.
nextKey	<code>const char* pDict-&gt;nextKey(rex_AttributeDictionary_t* pDict)</code>	Returns the name of the <i>next</i> attribute in the dictionary that follows the attribute returned in the last call to <b>firstKey</b> or <b>nextKey</b> .
nextKeyByType	<code>const abytes_t* pDict-&gt;nextKeyByType(rex_AttributeDictionary_t* pDict)</code>	Returns a pointer to the byte sequence defining the next attribute in the dictionary that follows the attribute returned in the last call to <b>firstKeyByType</b> or <b>nextKeyByType</b> .
put	<code>abool_t pDict-&gt;put(rex_AttributeDictionary_t* pDict, const char* &lt;pszAttribute&gt;, const char* &lt;pszValue&gt;, int &lt;iIndex&gt;)</code>	Converts <pszValue> to a sequence of bytes, according to the definition of <pszAttribute> in the server configuration. Associates that sequence of bytes with <pszAttribute> in the dictionary. When <iIndex> equals the special value <b>REX_REPLACE</b> , it replaces any existing instances of <pszAttribute> with a single value. When <iIndex> equals the special value <b>REX_APPEND</b> , it appends a new instance of <pszAttribute> to the end of the list of existing instances of <pszAttribute>. Otherwise, a new instance of <pszAttribute> is inserted at the position indicated. This method returns TRUE unless <pszAttribute> does not match any configured attributes or the value could not be converted to a legal value. When <iIndex> equals the special value <b>REX_AUGMENT</b> , only <b>put</b> <pszAttribute> when it does not already exist.

Table A-3 REX Attribute Dictionary Methods (continued)

Name	Syntax	Description
putBytes	<pre> abool_t pDict-&gt;putBytes(rex_AttributeDictionary_t* pDict, const char* &lt;pszAttribute&gt;, const abytes_t* &lt;pValue&gt;, int &lt;iIndex&gt;) </pre>	<p>Associates &lt;pValue&gt; with the attribute &lt;pszAttribute&gt; in the dictionary. When &lt;iIndex&gt; equals the special value <b>REX_REPLACE</b>, it replaces any existing instances of the &lt;pszAttribute&gt; with a single new value. When &lt;iIndex&gt; equals the special value <b>REX_APPEND</b>, it appends a new instance of &lt;pszAttribute&gt; to the end of the list of existing instances of &lt;pszAttribute&gt;. When &lt;iIndex&gt; equals the special value <b>REX_AUGMENT</b>, only put the &lt;pszAttribute&gt; when it does not already exist. Otherwise, a new instance of &lt;pszAttribute&gt; is inserted at the position indicated.</p> <p>This method returns TRUE unless the attribute name does not match any configured attributes.</p>
putBytesByType	<pre> abool_t pDict-&gt;putBytesByType(rex_AttributeDictionary_t* pDict, const abytes_t* &lt;pAttribute&gt;, const abytes_t* &lt;pValue&gt;, int &lt;iIndex&gt;) </pre>	<p>Associates &lt;pValue&gt; with the attribute &lt;pAttribute&gt; in the dictionary. When &lt;iIndex&gt; equals the special value <b>REX_REPLACE</b>, it replaces any existing instances of &lt;pAttribute&gt; with the new value. When &lt;iIndex&gt; equals the special value <b>REX_APPEND</b>, it appends a new instance of &lt;pAttribute&gt; to the end of the list of existing instances of &lt;pAttribute&gt;. When &lt;iIndex&gt; equals the special value <b>REX_AUGMENT</b>, only put &lt;pAttribute&gt; when it does not already exist. Otherwise, insert a new instance of &lt;pAttribute&gt; at the position indicated.</p> <p>This method returns TRUE unless the attribute name does not match any configured attributes.</p>
putByType	<pre> abool_t pDict-&gt;putByType(rex_AttributeDictionary_t* pDict, const abytes_t* &lt;pszAttribute&gt;, const char* &lt;pszValue&gt;, int &lt;iIndex&gt;) </pre>	<p>Converts &lt;pszValue&gt; to a sequence of bytes, according to the definition of &lt;pszAttribute&gt; in the server configuration. Associates that sequence of bytes with &lt;pszAttribute&gt; in the dictionary. When &lt;iIndex&gt; equals the special value <b>REX_REPLACE</b>, it replaces any existing instances of &lt;pszAttribute&gt; with a single new value. When &lt;iIndex&gt; equals the special value <b>REX_APPEND</b>, it appends a new instance of &lt;pszAttribute&gt; to the end of the list of existing instances of &lt;pszAttribute&gt;. Otherwise, it inserts a new instance of &lt;pszAttribute&gt; at the position indicated. This method returns TRUE unless &lt;pszAttribute&gt; does not match any configured attributes, or the value could not be converted to a legal value.</p>

Table A-3 REX Attribute Dictionary Methods (continued)

Name	Syntax	Description
remove	<pre>abool_t pDict-&gt;remove(rex_AttributeDictionary_t* pDict, const char* &lt;pszAttribute&gt;, int &lt;iIndex&gt;)</pre>	Removes the <pszAttribute> from the dictionary. When <iIndex> equals the special value <b>REX_REMOVE_ALL</b> , removes any existing instances of <pszAttribute>. Otherwise, it removes the instance of <pszAttribute> at the position indicated. Returns TRUE, even when the dictionary did not contain <pszAttribute> at the <iIndex>, unless <pszAttribute> does not match any configured attribute.
removeByType	<pre>abool_t pDict-&gt;removeByType(rex_AttributeDictionary_ t* pDict, const abytes_t* &lt;pAttribute&gt;, int &lt;iIndex&gt;)</pre>	Removes the <pAttribute> from the dictionary. When <iIndex> equals the special value <b>REX_REMOVE_ALL</b> , it removes any existing instances of <pszAttribute>. Otherwise, the instance of <pAttribute> at the position indicated is removed. Always returns TRUE, even when the dictionary did not contain <pAttribute> at the <iIndex>.
reschedule	<pre>abool_t pDict-&gt;reschedule(rex_AttributeDictionary_t* pDict)</pre>	Enables control over asynchronous activities. It enables you to collect similar activities and mark them as pending. You can then process them and reschedule them. You can only use this attribute with multithreaded services. Use caution when employing this method.
size	<pre>int pDict-&gt;size(rex_AttributeDictionary_t* pDict)</pre>	Returns the number of entries in the dictionary.
trace	<pre>abool_t pDict-&gt;trace(rex_AttributeDictionary_t* pDict, int &lt;iLevel&gt;, const char* &lt;pszFormat&gt;, ...)</pre>	Outputs a message into the packet tracing system used by the RADIUS server. At level 0, no tracing occurs. At level 1, only an indication the packet was received and a reply was sent is output. As the number gets higher, the amount of information output is greater, until at level 4, where everything traceable is output. The remaining arguments are formatted and sent to the tracing system at the specified level.

## REX Environment Dictionary

A dictionary is a data structure that contains key/value pairs. An Environment dictionary is a dictionary in which the keys and values are constrained to be strings. The REX Environment dictionary is used to communicate information from the script to the server and from script to script within the processing of a particular request. Note, there can be only one instance of a key in the Environment dictionary.

## REX Environment Dictionary Methods

The Environment dictionary uses active commands, called *methods*, to allow you to change and access the values in the dictionary. [Table A-4](#) lists all of the methods you can use with the REX Environment dictionary.

**Table A-4** REX Environment Dictionary Methods

Name	Syntax	Description
<b>allocateMemory</b>	<code>void* pDict-&gt;allocateMemory(rex_EnvironmentDictionary_t* pDict, unsigned int &lt;iSize&gt;)</code>	Allocate memory for use in scripts that persist only for the lifetime of this request. This memory is released when processing for this request is complete.
<b>clear</b>	<code>void pDict-&gt;clear(rex_EnvironmentDictionary_t* pDict)</code>	Removes all entries from the dictionary.
<b>containsKey</b>	<code>abool_t pDict-&gt;containsKey(rex_EnvironmentDictionary_t* pDict, const char* &lt;pszKey&gt;)</code>	Returns TRUE when the dictionary contains <pszKey>, otherwise returns FALSE.
<b>firstKey</b>	<code>const char* pDict-&gt;firstKey(rex_EnvironmentDictionary_t* pDict)</code>	Returns the name of the first key in the dictionary. Note, the keys are not stored sorted by name.
<b>get</b>	<code>const char* pDict-&gt;get(rex_EnvironmentDictionary_t* pDict, const char* &lt;pszKey&gt;)</code>	Returns the value associated with <pszKey> from the dictionary. When the dictionary does not contain <pszKey>, an empty string is returned.
<b>isEmpty</b>	<code>abool_t pDict-&gt;isEmpty(rex_EnvironmentDictionary_t* pDict)</code>	Returns TRUE when the dictionary has 0 entries, FALSE otherwise.
<b>log</b>	<code>abool_t pDict-&gt;log(rex_EnvironmentDictionary_t* pDict, int &lt;iLevel&gt;, const char* &lt;pszFormat&gt;, ...)</code>	Outputs a message into the logging system used by the RADIUS server. <iLevel> should be one of <b>REX_LOG_ERROR</b> , <b>REX_LOG_WARNING</b> , or <b>REX_LOG_INFO</b> . The <pszFormat> argument is treated as a <b>printf</b> -style format string, and it, along with the remaining arguments, are formatted and sent to the logging system at the specified level.
<b>nextKey</b>	<code>const char* pDict-&gt;nextKey(rex_EnvironmentDictionary_t* pDict)</code>	Returns the name of the next key in the dictionary that follows the key returned in the last call to <b>firstKey</b> or <b>nextKey</b> .
<b>put</b>	<code>abool_t pDict-&gt;put(rex_EnvironmentDictionary_t* pDict, const char* &lt;pszValue&gt;, const char* &lt;pszKey&gt;)</code>	Associates the value with <pszKey> in the dictionary, replacing any existing instance of <pszKey> with the new <pszValue>.
<b>remove</b>	<code>abool_t pDict-&gt;remove(rex_EnvironmentDictionary_t* pDict, const char* &lt;pszKey&gt;)</code>	Removes <pszKey> and the associated value from the dictionary. Always returns TRUE, even when the dictionary did not contain <pszKey>

Table A-4 REX Environment Dictionary Methods (continued)

Name	Syntax	Description
reschedule	<b>abool_t</b> <b>pDict-&gt;reschedule(rex_AttributeDictionary_t*</b> <b>pDict)</b>	Enables control over asynchronous activities. It enables you to collect similar activities and mark them as pending. You can then process them and reschedule them. You can only use this attribute with multithreaded services. Use caution when employing this method.
size	<b>int pDict-&gt;size(rex_EnvironmentDictionary_t*</b> <b>pDict)</b>	Returns the number of entries in the dictionary.
trace	<b>abool_t</b> <b>pDict-&gt;trace(rex_EnvironmentDictionary_t* pDict,</b> <b>int &lt;iLevel&gt;, const char* &lt;pszFormat&gt;, ...)</b>	Outputs a message into the packet tracing system used by the RADIUS server. At level 0, no tracing occurs. At level 1, only an indication the packet was received and a reply was sent is output. As the number gets higher, the amount of information output is greater, until at level 4, where everything traceable is output. The remaining arguments are formatted and sent to the tracing system at the specified level.

