Cisco Prime Access Registrar Tcl, REX and Java 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).

This appendix section also describes the following Java attribute dictionary:

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 Prime 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. This section contains the following topics:

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

Table A-1 Tcl Attribute Dictionary Methods (continued)

Name	Syntax	Description
nextKey	**$dict nextKey**	Returns the name of the next attribute in the dictionary that follows the attribute returned in the last call to **firstKey** or **nextKey**.
put	**$dict put** <*attribute*> <*value*> [<*index*>]	Associates <*value*> with the attribute <*attribute*> in the dictionary. When <*index*> is not provided or when <*index*> equals the special value **REPLACE**, any existing instances of <*attribute*> are replaced with the single value. When <*index*> is provided and equals the special value **APPEND**, 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 **AUGMENT**, only put the attribute when it does not already exist.
remove	**$dict remove** <*attribute*> [<*index*>]	Removes the <*attribute*> attribute from the dictionary. When <*index*> is not provided or when <*index*> equals the special value **REMOVE_ALL**, remove any existing instances of <*attribute*>. When <*index*> is provided and is a number, remove the instance of <*attribute*> at the position indicated. Always returns 1, even when the dictionary did not contain the <*attribute*> at that <*index*>.
size	**$dict size**	Returns the number of entries in the dictionary.
trace	**$dict trace** <*level*> <*message*> ...	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	**$dict clear**	Removes all entries from the dictionary.
containsKey	**$dict containsKey** <*key*>	Returns 1 when the dictionary contains the <*key*> key, otherwise returns 0.
firstKey	**$dict firstKey**	Returns the name of the first key in the dictionary. Note, the keys are not stored sorted by name.
get	**$dict get** <*key*>	Returns the value of <*key*> from the dictionary. When the dictionary does not contain the <*key*>, an empty string is returned.
isEmpty	**$dict isEmpty**	Returns 1 when the dictionary has no entries, otherwise returns 0.
log	**$dict log** <*level*> <*message*> …	Outputs a message into the logging system used by the RADIUS server. <*level*> should be one of **LOG_ERROR**, **LOG_WARNING**, or **LOG_INFO**. The remaining arguments are concatenated together and sent to the logging system at the specified level.
nextKey	**$dict nextKey**	Returns the name of the next key in the dictionary that follows the key returned in the last call to **firstKey** or **nextKey**.
put	**$dict put** <*key*> <*value*>	Associates <*value*> with the <*key*> key in the dictionary, replacing an existing instance of <*key*> with the new value.
remove	**$dict remove** <*key*>	Removes the <*key*> key from the dictionary. Always returns 1, even when the dictionary did not contain the <*key*>.
size	**$dict size**	Returns the number of entries in the dictionary.
trace	**$dict trace** <*level*> <*message*> …	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. This section contains the following topics:

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
addProfile	**abool_t pDict->addProfile(rex_AttributeDictionary_t* pDict, const char*** <*pszProfile*>**, int** <*iMode*>**)**	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 **REX_REPLACE**, it replaces any duplicate instances of the attributes in the dictionary with the attribute from the profile. When <*iMode*> equals the special value **REX_APPEND**, it appends a new instance of the attributes to any attributes already in the dictionary. When <*iMode*> equals the special value. When the mode is **REX_AUGMENT**, it adds the attribute in the dictionary, if it does not already exist in the dictionary.
allocateMemory	**void* pDict->allocateMemory(rex_AttributeDictionary_t* pDict, unsigned int** <*iSize*>**)**	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.

Table A-3 REX Attribute Dictionary Methods (continued)

Name	Syntax	Description
clear	**void pDict->clear(rex_AttributeDictionary_t* pDict)**	Removes all entries from the dictionary.
containsKey	**abool_t pDict->containsKey(rex_AttributeDictionary_t* pDict, const char*** *<pszAttribute>*)	Returns TRUE when the dictionary contains *<pszAttribute>*, otherwise returns FALSE.
containsKeyByType	**abool_t pDict->containsKeyByType(rex_AttributeDictionary_t* pDict, const abytes_t*** *<pAttribute>*)	Returns TRUE when the dictionary contains *<pAttribute>*, otherwise returns FALSE.
firstKey	**const char* pDict->firstKey(rex_AttributeDictionary_t* pDict)**	Returns the name of the first attribute in the dictionary. Note, the attributes are not stored in a sorted order of name.
firstKeyByType	**const abytes_t* pDict->firstKeyByType (rex_AttributeDictionary_t* pDict)**	Returns a pointer to the byte sequence defining the first attribute in the dictionary. Note, attributes are not stored sorted by name.
get	**const char* pDict->get(rex_AttributeDictionary_t* pDict, const char* pszAttribute, int <***iIndex***>, abool_t*** *<pbMore>*)	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 **get** 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 **get** should be made to retrieve other instances of the attribute.

Table A-3 REX Attribute Dictionary Methods (continued)

Name	Syntax	Description
getBytes	**const abytes_t* pDict->getBytes(rex_AttributeDictionary_t* pDict, const char* pszAttribute, int <***iIndex***>, abool_t*** <*pbMore*>**)**	Returns the value of the **<***iIndex***>**'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. When <*pbMore*> is non-zero, the **getBytes** 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 **getBytes** should be made to retrieve other instances of the attribute.
getBytesByType	**const abytes_t* pDict->getBytesByType (rex_AttributeDictionary_t* pDict, const abytes_t* pAttribute, int <***iIndex***>, abool_t*** <*pbMore*>**)**	Returns the value of the **<***iIndex***>**'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. When <*pbMore*> 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 **get** should be made to retrieve other instances of the attribute.
getByType	**const char* pDict->get(rex_AttributeDictionary_t* pDict,** const abytes_t* <*pszAttribute*>, int **<***iIndex***>**, abool_t* <*pbMore*>)	Returns the value of the **<***iIndex***>**'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. When <*pbMore*> is non-zero, the **getByType** 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 **getByType** should be made to retrieve other instances of the attribute.
getType	**const char* pDict->getByType(rex_AttributeDictionary_t* pDict, const abytes_t*** <*pAttribute*>**)**	Returns a pointer to the byte sequence defining the attribute, when the attribute name matches a configured attribute, zero otherwise.

Table A-3 REX Attribute Dictionary Methods (continued)

Name	Syntax	Description
isEmpty	**abool_t pDict->isEmpty(rex_AttributeDictionary_t* pDict)**	Returns TRUE when the dictionary has 0 entries, FALSE otherwise.
log	**abool_t pDict->log(rex_AttributeDictionary_t* pDict, int** <*iLevel*>**, const char*** <*pszFormat*>**, ...)**	Outputs a message into the logging system used by the RADIUS server. <*iLevel*> should be one of **REX_LOG_ERROR**, **REX_LOG_WARNING**, or **REX_LOG_INFO**. The **pszFormat** argument is treated as a **printf**-style format string, and it, along with the remaining arguments, are formatted and sent to the logging system at the specified level.
nextKey	**const char* pDict->nextKey(rex_AttributeDictionary_t* pDict)**	Returns the name of the *next* attribute in the dictionary that follows the attribute returned in the last call to **firstKey** or **nextKey**.
nextKeyByType	**const abytes_t* pDict->nextKeyByType(rex_AttributeDictionary_t* pDict)**	Returns a pointer to the byte sequence defining the next attribute in the dictionary that follows the attribute returned in the last call to **firstKeyByType** or **nextKeyByType**.
put	**abool_t pDict->put(rex_AttributeDictionary_t* pDict, const char*** <*pszAttribute*>**, const char*** <*pszValue*>**, int <***iIndex***>)**	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 **REX_REPLACE**, it replaces any existing instances of <*pszAttribute*> with a single value. When **<***iIndex***>** equals the special value **REX_APPEND**, 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 **REX_AUGMENT**, only **put** <*pszAttribute*> when it does not already exist.

Table A-3 REX Attribute Dictionary Methods (continued)

Name	Syntax	Description
putBytes	**abool_t pDict->putBytes(rex_AttributeDictionary_t* pDict, const char*** <*pszAttribute*>**, const abytes_t*** <*pValue*>**, int** <*iIndex*>**)**	Associates <*pValue*> with the attribute <*pszAttribute*> in the dictionary. When <*iIndex*> equals the special value **REX_REPLACE**, it replaces any existing instances of the <*pszAttribute*> with a single new value. When <*iIndex*> equals the special value **REX_APPEND**, it appends a new instance of <*pszAttribute*> to the end of the list of existing instances of <*pszAttribute*>. When <*iIndex*> equals the special value **REX_AUGMENT**, only put the <*pszAttribute*> when it does not already exist. Otherwise, a new instance of <*pszAttribute*> is inserted at the position indicated. This method returns TRUE unless the attribute name does not match any configured attributes.
putBytesByType	**abool_t pDict->putBytesByType(rex_AttributeDictionary_t* pDict, const abytes_t*** <*pAttribute*>**, const abytes_t*** <*pValue*>**, int** <*iIndex*>**)**	Associates <*pValue*> with the attribute <*pAttribute*> in the dictionary. When <*iIndex*> equals the special value **REX_REPLACE**, it replaces any existing instances of <*pAttribute*> with the new value. When <*iIndex*> equals the special value **REX_APPEND**, it appends a new instance of <*pAttribute*> to the end of the list of existing instances of <*pAttribute*>. When <*iIndex*> equals the special value **REX_AUGMENT**, only **put** <*pAttribute*> when it does not already exist. Otherwise, insert a new instance of <*pAttribute*> at the position indicated. This method returns TRUE unless the attribute name does not match any configured attributes.

Table A-3 REX Attribute Dictionary Methods (continued)

Name	Syntax	Description
putByType	**abool_t pDict->putByType(rex_AttributeDictionary_t* pDict, const abytes_t*** <*pszAttribute*>**, const char*** <*pszValue*>**, int** <*iIndex*>**)**	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 **REX_REPLACE**, it replaces any existing instances of <*pszAttribute*> with a single new value. When <*iIndex*> equals the special value **REX_APPEND**, it appends a new instance of <*pszAttribute*> to the end of the list of existing instances of <*pszAttribute*>. Otherwise, it inserts a new instance of <*pszAttribute*> 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.
remove	**abool_t pDict->remove(rex_AttributeDictionary_t* pDict, const char*** <*pszAttribute*>**, int** <*iIndex*>**)**	Removes the <*pszAttribute*> from the dictionary. When <*iIndex*> equals the special value **REX_REMOVE_ALL**, 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	**abool_t pDict->removeByType(rex_AttributeDictionary_t* pDict, const abytes_t*** <*pAttribute*>**, int** <*iIndex*>**)**	Removes the <*pAttribute*> from the dictionary. When <*iIndex*> equals the special value **REX_REMOVE_ALL**, 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	**abool_t pDict->reschedule(rex_AttributeDictionary_t* pDict)**	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.

Table A-3 REX Attribute Dictionary Methods (continued)

Name	Syntax	Description
size	**int pDict->size(rex_AttributeDictionary_t* pDict)**	Returns the number of entries in the dictionary.
trace	**abool_t pDict->trace(rex_AttributeDictionary_t* pDict, int** <*iLevel*>**, const char*** <*pszFormat*>**, ...)**	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
allocateMemory	**void* pDict->allocateMemory(rex_EnvironmentDictionary_t* pDict, unsigned int** <*iSize*>**)**	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.
clear	**void pDict->clear(rex_EnvironmentDictionary_t* pDict)**	Removes all entries from the dictionary.
containsKey	**abool_t pDict->containsKey(rex_EnvironmentDictionary_t* pDict, const char*** <*pszKey*>**)**	Returns TRUE when the dictionary contains <*pszKey*>, otherwise returns FALSE.
firstKey	**const char* pDict->firstKey(rex_EnvironmentDictionary_t* pDict)**	Returns the name of the first key in the dictionary. Note, the keys are not stored sorted by name.

Table A-4 REX Environment Dictionary Methods (continued)

Name	Syntax	Description
get	**const char* pDict->get(rex_EnvironmentDictionary_t* pDict, const char*** <*pszKey*>**)**	Returns the value associated with <*pszKey*> from the dictionary. When the dictionary does not contain <*pszKey*>, an empty string is returned.
isEmpty	**abool_t pDict->isEmpty(rex_EnvironmentDictionary_t* pDict)**	Returns TRUE when the dictionary has 0 entries, FALSE otherwise.
log	**abool_t pDict->log(rex_EnvironmentDictionary_t* pDict, int** <*iLevel*>**, const char*** <*pszFormat*>**, ...)**	Outputs a message into the logging system used by the RADIUS server. <*iLevel*> should be one of **REX_LOG_ERROR**, **REX_LOG_WARNING**, or **REX_LOG_INFO**. The <*pszFormat*> argument is treated as a **printf**-style format string, and it, along with the remaining arguments, are formatted and sent to the logging system at the specified level.
nextKey	**const char* pDict->nextKey(rex_EnvironmentDictionary_t* pDict)**	Returns the name of the next key in the dictionary that follows the key returned in the last call to **firstKey** or **nextKey**.
put	**abool_t pDict->put(rex_EnvironmentDictionary_t* pDict, const char*** <*pszValue*>**, const char*** <*pszKey*>**)**	Associates the value with <*pszKey*> in the dictionary, replacing any existing instance of <*pszKey*> with the new <*pszValue*>.
remove	**abool_t pDict->remove(rex_EnvironmentDictionary_t* pDict, const char*** <*pszKey*>**)**	Removes <*pszKey*> and the associated value from the dictionary. Always returns TRUE, even when the dictionary did not contain <*pszKey*>
reschedule	**abool_t pDict->reschedule(rex_AttributeDictionary_t* pDict)**	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.

Table A-4 REX Environment Dictionary Methods (continued)

Name	Syntax	Description
size	**int pDict->size(rex_EnvironmentDictionary_t* pDict)**	Returns the number of entries in the dictionary.
trace	**abool_t pDict->trace(rex_EnvironmentDictionary_t* pDict, int** <*iLevel*>**, const char*** <*pszFormat*>**, ...)**	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.

Java Attribute Dictionary

The AttributeDictionary is a dictionary of attributes, where the keys are the attribute types and the values are the data fields in the attribute. Both keys and values must conform to the definition of attributes in the server's Attribute Dictionary. Keys (types) can be either strings or byte arrays. If strings, they are the names of attributes. If byte arrays, they are the binary type. The type associated with a name can be retrieved by calling the static method getType(java.lang.String). Using byte arrays is slightly more efficient - methods that take String keys must do the mapping from String to byte array in the course of executing the method. Similiarly, values can be strings or byte arrays. Again, string values are converted to the appropriate binary representation when stored in an AttributeDictionary and back again when retrieved into a string variable.

Keys in an AttributeDictionary can be associated with multiple values. Each of the values associated with a key is ordered with an integer index denoting its position in the list of values. Given an AttributeDictionary, a key and an index, each value associated with a key can be looked up. This section contains the following topics:

Java Attribute Dictionary Methods

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

Table A-5 lists all of the methods you can use with the Request and Response dictionaries.

Table A-5 Java Attribute Dictionary Methods

Name	Syntax	Description
size	**public int size()**	Returns the number of distinct keys in the dictionary.
isEmpty	**public boolean isEmpty()**	Tests if the dictionary contains any entries.
clear	**public void clear()**	Removes all entries from the dictionary.
containsKey	**public boolean containsKey(java.lang.String key)**	Returns true if an entry exists for key.
get	**public java.lang.String get(java.lang.String key)**	Returns the first value associated with the key.
get	**public java.lang.String get(java.lang.String key, int index)**	Returns the value at position index associated with the key.
put	**public boolean put(java.lang.String key, java.lang.String value)**	Associates key with a value. Any existing values associated with the key are removed before adding this association.
put	**public boolean put(java.lang.String key, java.lang.String value, int index)**	Associates key with a value depending on the value of index. If index equals **Extension.EXT_REPLACE**, any existing values are removed before adding this new association. If index equals **Extension.EXT_APPEND**, a new value is added at the end of the list of existing values. If index equals **Extension.EXT_AUGMENT**, the new association is only made if the dictionary does not already have an entry for key. If index is a number greater than or equal to 0 and less than the number of entries in the list, the value is inserted at that position in the list. Otherwise, the value is appended at the end of the list.
getBytes	**public byte[] getBytes(java.lang.String key)**	Returns the first value associated with the key.
getBytes	**public byte[] getBytes(java.lang.String key, int index)**	Returns the value at position index associated with key.
putBytes	**public boolean putBytes(java.lang.String key, byte[] value)**	Associates key with value. Any existing values associated with key are removed before adding this association.

Table A-5 Java Attribute Dictionary Methods (continued)

Name	Syntax	Description
putBytes	**public boolean putBytes(java.lang.String key, byte[] value, int index)**	Associates key with a value depending on the value of index. If index equals **Extension.EXT_REPLACE**, any existing values are removed before adding this new association. If index equals **Extension.EXT_APPEND**, a new value is added at the end of the list of existing values. If index equals **Extension.EXT_AUGMENT**, the new association is only made if the dictionary does not already have an entry for key. If index is a number greater than or equal to 0 and less than the number of entries in the list, the value is inserted at that position in the list. Otherwise, the value is appended at the end of the list.
remove	**public void remove(java.lang.String key)**	Removes key (and all corresponding values) from the dictionary. This method does nothing if key is not in the dictionary.
remove	**public void remove(java.lang.String key, int index)**	Removes value at the position index that is associated with key. If the index equals **Extension.EXT_REMOVE_ALL** or if the value being removed is the last value associated with key, the key is removed from the dictionary. This method does nothing if key is not in the dictionary.
addProfile	**public boolean addProfile(java.lang.String profileName)**	Adds all the attributes contained in the specified profile into the dictionary. Any existing attributes that have the same keys as attributes in the profile are removed before adding the new attributes.
Addprofile	**boolean addProfile(java.lang.String profileName, int mode)**	Adds all the attributes contained in the specified profile into the dictionary. Any existing attributes that have the same keys as attributes in the Profile will be treated depending on the mode value. For each attribute in the Profile, if mode equals **Extension.EXT_REPLACE**, any values associated with the attribute in the dictionary are removed before adding the attribute. If index equals **Extension.EXT_APPEND**, a new value is added at the end of the list of existing values. If index equals **Extension.EXT_AUGMENT**, a new value is added only if the dictionary does not already have an entry for the given key.
getType	**public static byte[] getType(java.lang.String key)**	Takes the name of the attribute (as a string) and returns the binary form of key.

Table A-5 Java Attribute Dictionary Methods (continued)

Name	Syntax	Description
keys	**public java.util.Enumeration keys()**	Returns an enumeration of the keys in the dictionary. The general contract for the keys method is that an Enumeration object is returned that will generate all the keys for which the dictionary contains entries.
elements	**public java.util.Enumeration elements()**	Returns an enumeration of the entries in the dictionary. The general contract for the elements method is that an Enumeration object is returned that will generate all the elements contained in entries in the dictionary. Keys with multiple values will result in multiple elements being returned.
keysByType	**public java.util.Enumeration keysByType()**	Returns an enumeration of the keys in the dictionary. The general contract for the keys method is that an Enumeration object is returned that will generate all the keys for which the dictionary contains entries.

Java Environment Dictionary

The Environment Dictionary can be used to store information between Extensions invoked subsequently on a given request or can be used to pass information between the Extension and the server properly.

The Environment Dictionary maps keys to values, where the keys and values are strings. In any one instance of the Environment Dictionary, every key is associated with at most one value. Given an Environment Dictionary and a key, the associated value can be looked up. Any non-null string can be used as a key and value.

Java 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-6 lists all of the methods you can use with the java Environment dictionary.

Table A-6 Java Environment Dictionary Methods

Name	Syntax	Description
size	**public int size()**	Returns the number of entries (distinct keys) in the dictionary.
isEmpty	**public boolean isEmpty()**	Tests if the dictionary contains no entries.
clear	**public void clear()**	Removes all entries from the dictionary.
containsKey	**public boolean containsKey(java.lang.String key)**	Returns true if the dictionary contains an entry for key.
get	**public java.lang.String get(java.lang.String key)**	Returns the value associated with key in the dictionary.

Table A-6 Java Environment Dictionary Methods (continued)

Name	Syntax	Description
put	**public boolean put(java.lang.String key, java.lang.String value)**	Associates key with value.
remove	**public void remove(java.lang.String key)**	Removes key (and its corresponding value) from this dictionary. This method does nothing if key is not in the dictionary.
keys	**public java.util.Enumeration keys()**	Returns an enumeration of the keys in the dictionary. The general contract for the keys method is that an Enumeration object is returned that will generate all the keys for which the dictionary contains entries.
elements	**public java.util.Enumeration elements()**	Returns an enumeration of the entries in the dictionary. The general contract for the elements method is that an Enumeration object is returned that will generate all the elements contained in entries in the dictionary.
log	**public static void log(int level,java.lang.String message)**	Prints a message in the server log at the specified level.
trace	**public void trace(int level, java.lang.String message)**	Prints a message in the server trace file at the specified level.
reschedule	**public void reschedule()**	Informs the server that it should take back ownership of the request associated with the dictionary and continue processing it.

Interface Extension

Classes that are going to be used as scripts or services from Access Registrar must implement the Extension interface. When a Java scripting point or service is encountered during the processing of a request, the server will call the runExtension method defined in this interface and implemented by the appropriate class.

Interface Extension Methods

Table A-7 lists the methods you can use for interface extension

Table A-7 Interface Extension Methods

Name	Syntax	Description
runExtension	**int runExtension(int iExtensionPoint, AttributeDictionary request, AttributeDictionary response,EnvironmentDictionary environment)**	This method is called whenever a Java scripting point or service is encountered during the processing of a request. When runExtension is used as a script, it should process requests as quickly as possible, without blocking. This is because the server has a limited number of threads that it is using to process requests and if any one extension takes too long to run, it is likely that many requests will be delayed as each one calls the extension. runExtension must return either **EXT_OK** to indicate that processing of this request should continue or **EXT_ERROR** to indicate that an error occurred while processing this request and that the request should be dropped. Extensions should always log an error before returning **EXT_ERROR** so that the administrator has a way to determine the problem that was encountered. When runExtension is used as a service, it will be called once before requests start coming in (with the iExtensionPoint parameter set to **EXT_START_SERVICE**) to give the extension the opportunity to initialize resources needed to process requests, and once after the last request has been received (with the iExtensionPoint parameter set to **EXT_STOP_SERVICE**) to give the extension the opportunity to release those resources before stopping. runExtension must return one of the following values: **EXT_OK, EXT_ERROR** or **EXT_PENDING. EXT_PENDING** should be returned to inform the server that the extension has taken ownership of the request, will process the request on a background thread, and will inform the server when it is time to continue processing the request by calling reschedule() on one of the request's dictionaries.

Interface ExtensionforSession

Classes that are going to be used as scripts at Session Manager level from Cisco Prime Access Registrar must implement the ExtensionForSession interface. When a Java scripting point or service is encountered during the processing of a request, the server will call the runExtension method defined in this interface and implemented by the appropriate class.

Interface Extensionforsession Methods

Table A-8 lists the methods you can use for interface extensionforsession

Table A-8 Interface Extensionforsession Methods

Name	Syntax	Description
runExtension	**int runExtension(int iExtensionPoint, AttributeDictionary request, AttributeDictionary response, EnvironmentDictionary environment, SessionRecord session)**	This method is called whenever a Java scripting point or service is encountered during the processing of a request. When runExtension is used as a script, it should process requests as quickly as possible, without blocking. This is because the server has a limited number of threads that it is using to process requests and if any one extension takes too long to run, it is likely that many requests will be delayed as each one calls the extension. runExtension must return either **EXT_OK** to indicate that processing of this request should continue or **EXT_ERROR** to indicate that an error occurred while processing this request and that the request should be dropped. Extensions should always log an error before returning **EXT_ERROR** so that the administrator has a way to determine the problem that was encountered. When runExtension is used as a service, it will be called once before requests start coming in (with the iExtensionPoint parameter set to **EXT_START_SERVICE**) to give the extension the opportunity to initialize resources needed to process requests, and once after the last request has been received (with the iExtensionPoint parameter set to **EXT_STOP_SERVICE**) to give the extension the opportunity to release those resources before stopping. runExtension must return one of the following values: **EXT_OK, EXT_ERROR** or **EXT_PENDING**. **EXT_PENDING** should be returned to inform the server that the extension has taken ownership of the request, will process the request on a background thread, and will inform the server when it is time to continue processing the request by calling reschedule() on one of the request's dictionaries.

Interface Extensionwithinitialization

Classes that are going to be used as scripts or services from Access Registrar implements the ExtensionWithInitialization interface. ExtensionWithInitialization extends the Extension interface with methods to initialize and destroy the extension. initialize(java.lang.String) is called when the extension is first loaded, with the string argument being set from the InitializeArg property that was defined in the server configuration when the extension was defined (either as a Script or a Service). Destroy() is called before the extension is unloaded.

Interface Extensionwithinitialization Methods

Table A-9 lists the methods you can use for Interface Extensionwithinitialization.

Table A-9 Interface Extensionwithinitialization Methods

Name	Syntax	Description
initialize	**void initialize(java.lang.String initializeArg)**	This method is called by the server when the extension is first loaded.
destroy	**void destroy()**	This method is called by the server when the extension is going to be unloaded.

Interface ExtensionforSessionwithinitialization

Classes that are going to be used as scripts from Access Registrar at Session Manager level implement the ExtensionForSessionWithInitialization interface. ExtensionForSessionWithInitialization extends the ExtensionForSession interface with methods to initialize and destroy the extension. initialize(java.lang.String) is called when the extension is first loaded, with the string argument being set from the InitializeArg property that was defined in the server configuration when the extension was defined (either as a script or a service). Destroy () is called before the extension is unloaded.

Interface Extensionforsessionwithinitialization Methods

Table A-10 lists the methods you can use for Interface Extensionforsessionwithinitialization.

Table A-10 Interface Extensionforsessionwithinitialization Methods

Name	Syntax	Description
initialize	**void initialize(java.lang.String initializeArg)**	This method is called by the server when the extension is first loaded.
destroy	**void destroy()**	This method is called by the server when the extension is going to be unloaded.

Interface MarkerExtension

This is just going to be a marker interface containing various member variables which can be used in interfaces/classes extending from this interface. Extension and ExtensionForSession interfaces will extend this interface.

Variables in the Marker Extension Interface

Table A-11 lists the variables in the marker extension interface.

Table A-11 Marker Extension Interface Variables

Name	Syntax	Description
EXT_OK	**static final int EXT_OK**	Returns **EXT_OK** by implementation of runExtension() to indicate that the extension operated correctly and processing of the request should continue.
EXT_ERROR	**static final int EXT_ERROR**	Returns **EXT_ERROR** by implementation of runExtension() to indicate that the extension failed in some way and processing of the request should NOT continue.
EXT_PENDING	**static final int EXT_PENDING**	Returns **EXT_PENDING** by implementations of runExtension() to indicate that the extension operated correctly and the extension wants to take ownership of the request for a while. Further processing of the request by the server will be postponed until the extension indicates that it can do so by calling the reschedule method on any of the dictionaries.
EXT_LOG_ERROR	**static final int EXT_LOG_ERROR**	Indicates that the message should be logged with a severity of ERROR, when passed to log() in the level parameter.
EXT_LOG_WARNING	**static final int EXT_LOG_WARNING**	Indicates that the message should be logged with a severity of WARNING, when passed to log() in the level parameter.
EXT_LOG_INFO	**static final int EXT_LOG_INFO**	Indicates that the message should be logged with a severity of INFO, when passed to log() in the level parameter.
EXT_REMOVE_ALL	**static final int EXT_REMOVE_ALL**	Indicates that all values associated with the specified key should be removed, when passed to AttributeDictionary::remove() in the index parameter.
EXT_REPLACE	**static final int EXT_REPLACE**	Indicates that all existing values associated with the specified key(s) should be removed before adding the new value(s), when passed to AttributeDictionary::put() (and its variants) in the index parameter or to AttributeDictionary::addProfile() in the mode parameter.
EXT_APPEND	**static final int EXT_APPEND**	Indicates that the new value(s) should be appended to the end of the list of any existing values associated with the specified key(s), when passed to AttributeDictionary::put() (and its variants) in the index parameter or to AttributeDictionary::addProfile() in the mode parameter.

Table A-11 Marker Extension Interface Variables (continued)

Name	Syntax	Description
EXT_AUGMENT	**static final int EXT_AUGMENT**	Indicates that the new association(s) should only be added if the dictionary does not already have an entry for the given key(s), when passed to AttributeDictionary::put() (and its variants) in the index parameter or to AttributeDictionary::addProfile() in the mode parameter.
EXT_START_SERVICE	**static final int EXT_START_SERVICE**	Indicates that the extension should do whatever is necessary to prepare to offer service, when passed to extensions used as services. This may include starting background threads, opening database connections, and so on.
EXT_AUTHENTICATION_SERVICE	**static final int EXT_AUTHENTICATION_SERVICE**	Indicates that the extension should authenticate the current request, when passed to extensions used as services. To indicate whether the request was authenticated or not, the extension should set the EnvironmentDictionary entry for "Response-Type" to either "Access-Accept" or "Access-Reject".
EXT_AUTHORIZATION_SERVICE	**static final int EXT_AUTHORIZATION_SERVICE**	Indicates that the extension should authorize the current request, when passed to extensions used as services.
EXT_AUTHENTICATION_AND_AUTHORIZATION_SERVICE	**static final int EXT_AUTHENTICATION_AND_AUTHORIZATION_SERVICE**	Indicates that the extension should both authenticate and authorize the current request, when passed to extensions used as services. To indicate whether the request was authenticated or not, the extension should set the EnvironmentDictionary entry for "Response-Type" to either "Access-Accept" or "Access-Reject".
EXT_ACCOUNTING_SERVICE	**static final int EXT_ACCOUNTING_SERVICE**	Indicates that the extension should produce an accounting record for the current request, when passed to extensions used as services.
EXT_STOP_SERVICE	**static final int EXT_STOP_SERVICE**	Indicates that the extension should do whatever is necessary to shut down, when passed to extensions used as services. This may include stopping background threads, closing database connections and so on.
EXT_NAS_STARTED_ACCOUNTING_SERVICE	**static final int EXT_NAS_STARTED_ACCOUNTING_SERVICE**	Indicates that the NAS identified in the EnvironmentDictionary (by either the "NAS-Identifier" or "NAS-IP-Address" entries) has indicated that it is starting up, when passed to extensions used as services. This may be used by extensions to prepare to receive requests from this particular NAS if the extension treats requests from different NASs differently.

Table A-11 Marker Extension Interface Variables (continued)

Name	Syntax	Description
EXT_NAS_STOPPED_ACCOUNTING_SERVICE	**static final int EXT_NAS_STOPPED_ACCOUNTING_SERVICE**	Indicates that the NAS identified in the EnvironmentDictionary (by either the "NAS-Identifier" or "NAS-IP-Address" entries) has indicated that it is shutting down, when passed to extensions used as services. This may be used by extensions to recover any resources associated with this NAS if the extension treats requests from different NASs differently.
EXT_INCOMING_SERVER_SCRIPTING_POINT	**static final int EXT_INCOMING_SERVER_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/IncomingScript**, when passed to extensions used as scripts.
EXT_INCOMING_VENDOR_SCRIPTING_POINT	**static final int EXT_INCOMING_VENDOR_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/Vendors/**<*vendor*>**/IncomingScript**. when passed to extensions used as scripts.
EXT_INCOMING_CLIENT_SCRIPTING_POINT	**static final int EXT_INCOMING_CLIENT_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/Clients/**<*client*>**/IncomingScript** or from the script **/Radius/RemoteServers/**<*server*>**/IncomingScript**, when passed to extensions used as scripts.
EXT_INCOMING_SERVICE_SCRIPTING_POINT	**static final int EXT_INCOMING_SERVICE_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/Services/**<*service*>**/IncomingScript**, when passed to extensions used as scripts.
EXT_USERGROUP_AUTHENTICATION_SCRIPTING_POINT	**static final int EXT_USERGROUP_AUTHENTICATION_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/UserGroups/**<*group*>**/AuthenticationScript**, when passed to extensions used as scripts.
EXT_USERRECORD_AUTHENTICATION_SCRIPTING_POINT	**static final int EXT_USERRECORD_AUTHENTICATION_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/UserLists/**<*userlist*>**/**<*user*>**/AuthenticationScript**, when passed to extensions used as scripts.
EXT_USERGROUP_AUTHORIZATION_SCRIPTING_POINT	**static final int EXT_USERGROUP_AUTHORIZATION_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/UserGroups/**<*group*>**/AuthorizationScript**, when passed to extensions used as scripts.
EXT_USERRECORD_AUTHORIZATION_SCRIPTING_POINT	**static final int EXT_USERRECORD_AUTHORIZATION_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/UserLists/**<*userlist*>**/**<*user*>**/AuthorizationScript**, when passed to extensions used as scripts.
EXT_OUTGOING_SERVICE_SCRIPTING_POINT	**static final int EXT_OUTGOING_SERVICE_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/Services/**<*service*>**/OutgoingScript**, when passed to extensions used as scripts.

Table A-11 Marker Extension Interface Variables (continued)

Name	Syntax	Description
EXT_OUTGOING_CLIENT_SCRIPTING_POINT	**static final int EXT_OUTGOING_CLIENT_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/Clients/**<*client*>**/OutgoingScript** or from the script **/Radius/RemoteServers/**<*server*>**/OutgoingScript**, when passed to extensions used as scripts.
EXT_OUTGOING_VENDOR_SCRIPTING_POINT	**static final int EXT_OUTGOING_VENDOR_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/Vendors/**<*vendor*>**/OutgoingScript**. when passed to extensions used as scripts.
EXT_OUTGOING_SERVER_SCRIPTING_POINT	**static final int EXT_OUTGOING_SERVER_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/OutgoingScript**, when passed to extensions used as scripts.
EXT_REMOTE_SERVER_OUTAGE_SCRIPTING_POINT	**static final int EXT_REMOTE_SERVER_OUTAGE_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/Services/**<*service*>**/OutageScript**, when passed to extensions used as scripts.
EXT_INCOMING_SESSIONMANAGER_SCRIPTING_POINT	**static final int EXT_INCOMING_SESSIONMANAGER_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/SessionManagers/**<*sessionmgr*>**/Incoming Script**, when passed to extensions used as scripts.
EXT_OUTGOING_SESSIONMANAGER_SCRIPTING_POINT	**static final int EXT_OUTGOING_SESSIONMANAGER_SCRIPTING_POINT**	Indicates that the extension is being called from the script **/Radius/SessionManagers/**<*sessionmgr*>**/Outgoing Script**, when passed to extensions used as scripts.

Class Sessionrecord

Each request processed by an Extension will have a corresponding session. The methods present in this class operate on the attributes cached in that session record. Group of attributes are cached as an AttributeDictionary in the session record.

Session Record Methods

Table A-12 lists the methods you can use for Session record.

Table A-12 Session Record Methods

Name	Syntax	Description
get	**public java.lang.String get(java.lang.String key)**	Returns the first value associated with key.
get	**public java.lang.String get(java.lang.String key,int index)**	Returns the value at position index associated with key.

Table A-12 Session Record Methods (continued)

Name	Syntax	Description
put	**public boolean put(java.lang.String key,java.lang.String value)**	Associates key with value and stores it to the session record. Any existing values associated with key are removed before adding this association. The value can be retrieved by calling the get method with a key that is equal to the original key.
put	**public boolean put(java.lang.String key,java.lang.String value, int index)**	Associates key with value depending on the value of index and stores it in the session record. If index equals **ExtensionForSession.EXT_REPLACE**, any existing values are removed before adding this new association. If index equals **ExtensionForSession.EXT_APPEND**, the new value is added at the end of the list of existing values. If index equals **ExtensionForSession.EXT_AUGMENT**, the new association is only made if the session record does not already have an entry for key. If index is a number greater than or equal to 0 and less than the number of entries in the list, the value is inserted at that position in the list. Otherwise, the value is appended at the end of the list. The value can be retrieved by calling the get method with a key that is equal to the original key and the appropriate index.
remove	**public boolean remove(java.lang.String key)**	Removes key (and all corresponding values) from the session record. This method does nothing if key is not in the session record.
remove	**public boolean remove(java.lang.String key, int index)**	Removes value at the position index that is associated with key. If the index equals **ExtensionForSession.EXT_REMOVE_ALL** or if the value being removed is the last value associated with key, the key is removed from the session record. This method does nothing if key is not in the session record.
getSessionInfo	**public java.lang.String getSessionInfo()**	Returns Session-ID, Session-Start-Time and Session-Last-Accessed-Time of the session record.

Note A sample java script is available in the following path "/cisco-ar/examples/java" after the installation of AR.