HP Service Manager

For the supported Windows® and UNIX® operating systems

Software Version: 9.31

Programming Guide

Document Release Date: October 2012

Software Release Date: October 2012

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© Copyright 1994-2012 Hewlett-Packard Development Company, L.P.

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Java is a registered trademark of Oracle and/or its affiliates.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Oracle® is a registered US trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of The Open Group.

For a complete list of open source and third party acknowledgements, visit the HP Software Support Online web site and search for the product manual called *HP Service Manager Open Source and Third Party License Agreements*.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Online web site at:

http://www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

Programming Guide 1

Contents 5

Overview 19

System language 20

Data types 20

List: Data types 20

Data type: Primitive 21

Data type: Number 22

Data type: Character 22

Data type: Time 23

Data type: Boolean 24

Data type: Label 24

Data type: Compound 25

Data type: File/Record 25

File variables 25

Data type: Offset 26

Data type: Array 26

Data type: Structure 27

Data type: Operator 28

Arithmetic operators 30

Logical operators 31

Relational operators 31

Data Type: Expression 31

Data type: Pseudo field 32

Data type: Global variable 32

Data type: Local variable 32

Reserved words 33

Literals 33
Variables 33
Creating variables 34
Global variables 34
List: Default variables 35
Variable pools 37
Statements 38
Assignment statements 40
List: Command line calls 40
Generic command line calls 40
Direct command line calls 40
Call a RAD application 50
Locate a RAD function 51
RAD Debugger 52
Start RAD Debugger 52
RAD Debugger commands 52
Introduction to JavaScript in Service Manager 55
Language overview 55
Learning JavaScript 55
Basic rules of JavaScript 56
Structure of a JavaScript application 56
Expressions and operators 57
JavaScript objects and properties 57
Built-in objects and functions 57
Conditional statements 57
While and loop statements 58
Break and continue statements 59
Exception handling statements 60
Where can I use JavaScript in Service Manager? 61
What is the ScriptLibrary? 61
ScriptLibrary editing environment 62
Calling a function in the ScriptLibrary 62

Example: Function to control close button label in Change Management 64
The log4js script 64
API Details 65
Performance impact 66
Examples of calling JavaScript from different Service Manager tools 66
Format Control 66
Links 67
Trigger scripts 69
Triggers example 69
Using RAD functions in JavaScript 71
JavaScript function: parse_evaluate() 71
RAD functions: parse() and evaluate() 72
JavaScript global properties 72
Query operators 72
JavaScript return code properties 73
Order of execution of JavaScript versus RAD language 74
Corresponding JavaScript syntax for RAD language tasks 76
System Language reference 77
Deleted RAD applications 77
List: RAD functions 79
RAD function: add.graphnodes(xml) 85
RAD function: cache.flush 85
RAD function: cleanup 86
RAD function: contents 87
RAD function: copyright 87
RAD function: currec 87
RAD function: current.device 88
RAD function: current.format 88
RAD function: cursor.field.contents 88
RAD function: cursor.field.name.set 88
RAD function: cursor.field.name 89
RAD function: cursor.field.readonly 89

RAD function: cursor.line 90
RAD function: date 90
RAD function: day 90
RAD function: dayofweek 91
RAD function: dayofyear 91
RAD function: delete 92
RAD function: denull 92
RAD function: descriptor 93
RAD function: evaluate 93
RAD function: evaluate.query 94
RAD function: exists 94
RAD function: fduplicate 95
RAD function: filename 95
RAD function: frestore 96
RAD function: genout 96
RAD function: get.base.form.name 97
RAD function: get.dateformat 97
RAD function: get.graph.action 98
RAD function: get.graph.id 98
RAD function: get.graph.target 98
RAD function: get.graphnode.id 98
RAD function: get.lock.owner 99
RAD function: get.timezoneoffset 99
RAD function: get.uid 99
RAD function: gui 100
RAD function: index 100
RAD function: insert 101
RAD function: iscurrent 102
RAD function: jscall 103
RAD function: lioption 103
RAD function: lng 105
RAD function: locks 106

RAD function: logoff 107
RAD function: logon 107
RAD function: mandant 107
RAD function: max 108
RAD function: messages 109
RAD function: min 111
RAD function: modtime 111
RAD function: month 112
RAD function: multiselect.selection("fieldcontents") 112
RAD function: multiselect.selection("fieldname") 113
RAD function: multiselect.selection("rows") 113
RAD function: multiselect.selection("selected") 114
RAD function: multiselect.selection("selections") 114
RAD function: multiselect.selection("tablename") 114
RAD function: null 114
RAD function: nullsub 115
RAD function: operator 115
RAD function: option 116
RAD function: parse 116
RAD function: perf 117
RAD function: printer 117
RAD function: policyread 118
RAD function: processes 120
RAD function: prof 121
RAD function: recordcopy 122
RAD function: recordtostring 123
RAD function: round 124
RAD function: same 124
RAD function: scmsg 125
RAD function: set.timezone 126
RAD function: setsort 126
RAD function: shutdown 127

RAD function: simple.file.load 127
RAD function: str 128
RAD function: stradj 128
RAD function: strchrcp 129
RAD function: strchrin 129
RAD function: strclpl 130
RAD function: strclpr 131
RAD function: strcpy 131
RAD function: strdel 132
RAD function: strins 132
RAD function: strpadl 133
RAD function: strpadr 134
RAD function: strrep 134
RAD function: strtrml 135
RAD function: strtrmr 135
RAD function: substr 136
RAD function: substrb 136
RAD function: sysinfo.get("ActiveFloatUsers") 137
RAD function: sysinfo.get("ActiveLicenses") 138
RAD function: sysinfo.get("ActiveNamedUsers") 138
RAD function: sysinfo.get("AuthMode") 138
RAD function: sysinfo.get("ClientNetAddress") 138
RAD function: sysinfo.get("ClientOSName") 139
RAD function: sysinfo.get("ClientPID") 139
RAD function: sysinfo.get("ClientVersion") 139
RAD function: sysinfo.get("Display") 139
RAD function: sysinfo.get("Environment") 140
RAD function: sysinfo.get("languagecode") 140
RAD function: sysinfo.get("MaxFloatUsers") 140
RAD function: sysinfo.get("MaxLicenses") 141
RAD function: sysinfo.get("Mode") 141
RAD function: sysinfo.get("PrevLabel") 141

RAD function: sysinfo.get("PrintOption") 142
RAD function: sysinfo.get("Quiesce") 142
RAD function: sysinfo.get("RecList") 142
RAD function: sysinfo.get("ServerNetAddress") 142
RAD function: sysinfo.get("ServerNetPort") 142
RAD function: sysinfo.get("ServerPID") 143
RAD function: sysinfo.get("Telephony") 143
RAD function: sysinfo.get("ThreadID") 143
RAD function: sysinfo.get("TotalFloatUsers") 144
RAD function: sysinfo.get("TotalLicenses") 144
RAD function: sysinfo.get("TotalNamedUsers") 144
RAD function: sysinfo.get("TotalProcs") 144
RAD function: sysinfo.get("TotalSystemProcs") 144
RAD function: sysinfo.get("TotalUserProcs") 145
RAD function: time 145
RAD function: tod 145
RAD function: tolower 146
RAD function: toupper 146
RAD function: translate 146
RAD function: trunc 147
RAD function: type 148
RAD function: val 148
RAD function: version 150
RAD function: year 150
List: rtecalls() 151
rtecall("alalnum") function 153
rtecall("counter") function 153
rtecall("datemake") function 154
rtecall("escstr") function 155
rtecall("filecopy") function 156
rtecall("fileinit") function 157
rtecall("filldate") function 158

rtecall("getnumber") function 158
rtecall("getrecord") function 159
rtecall("getunique") function 160
rtecall("isalnum") function 161
rtecall("isalpha") function 162
rtecall("islower") function 163
rtecall("isnumeric") function 163
rtecall("isupper") function 164
rtecall("log") function 165
rtecall("passchange") function 166
rtecall("policycheck") function 167
rtecall("qbeform") function 167
rtecall("recdupl") function 168
rtecall("recordexists") function 169
rtecall("refresh") function 169
rtecall("rfirst") function 170
rtecall("rgoto") function 170
rtecall("rid") function 171
rtecall("scantable") function 172
rtecall("sort") function 173
rtecall("statusupdate") function 175
rtecall("transtart") function 175
rtecall("transtop") function 176
rtecall("trigger") function 177
List: RAD routines 177
RAD routine: as.copy 179
RAD routine: as.delete 180
RAD routine: as.get.name 181
RAD routine: as.insert 181
RAD routine: as.move 182
RAD routine: as.options 183
RAD routine: as.sort 184

RAD routine: axces.apm 185
RAD routine: axces.cm3.cts.write 186
RAD routine: axces.cm3 186
RAD routine: axces.database 187
RAD routine: axces.email 188
RAD routine: axces.email.receive 188
RAD routine: axces.fax 189
RAD routine: axces.move.intoout 189
RAD routine: axces.page 190
RAD routine: axces.rm 191
RAD routine: axces.sap.hybrid.evin 191
RAD routine: axces.sm 192
RAD routine: axces.software 192
RAD routine: axces.write 193
RAD routine: database 194
RAD routine: es.approval 194
RAD routine: fill.fc 195
RAD routine: getnumb.fc 195
RAD routine: marquee.publish 196
RAD routine: marquee.send 197
RAD routine: message.fc 197
RAD routine: post.fc 199
RAD routine: publish 199
RAD routine: query.stored 200
RAD routine: scauto.inventory 200
RAD routine: sort.array 201
RAD routine: validate.fields 201
RAD routine: wmi.inventory.check 202
JavaScript and Service Manager reference 203
JavaScript global System objects 203
JavaScript object: system.files 204
JavaScript object: system.forms 204

JavaScript object: system.functions 205
JavaScript object: system.functions.scmsg 206
JavaScript object: system.library 207
JavaScript object: system.oldrecord 208
JavaScript object: system.record 208
JavaScript object: system.sysinfo 209
JavaScript object: system.threads 209
JavaScript object: system.user 210
JavaScript object: system.users 210
JavaScript object: system.vars 211
List: JavaScript global methods 212
JavaScript global method: base64Decode 212
JavaScript global method: base64Encode 213
JavaScript global method: compile 214
JavaScript global method: doHTTPRequest 215
JavaScript global method: doSOAPRequest 216
JavaScript global method: execute 219
JavaScript global method: help 220
JavaScript global method: makeSCWebURL 220
JavaScript global method: print 222
JavaScript global method: Quit 223
JavaScript global method: RCtoString 224
JavaScript global method: readFile 225
JavaScript global method: setAppMessage 226
JavaScript global method: stripHtml 227
JavaScript global method: uncompressFile 227
JavaScript global method: writeAttachmentToFile 228
JavaScript global method: writeFile 229
JavaScript global method: xmlstring 231
Service Manager defined JavaScript objects 231
JavaScript object: Attachment 232
JavaScript object: Datum 234

JavaScript object: Header 236
JavaScript object: SCDatum 237
JavaScript object: SCFile 239
JavaScript method: SCFile.deleteAttachment(attachID) 242
JavaScript method: SCFile.deleteAttachments() 243
JavaScript method: SCFile.doAction() 244
JavaScript method: SCFile.doCount() 246
JavaScript method: SCFile.doDelete() 247
JavaScript method: SCFile.doInsert() 248
JavaScript method: SCFile.doPurge() 249
JavaScript method: SCFile.doRemove() 251
JavaScript method: SCFile.doSave() 253
JavaScript method: SCFile.doSelect() 254
JavaScript method: SCFile.doUpdate() 255
JavaScript method: SCFile.getAttachment(attachID) 256
JavaScript method: SCFile.getAttachments() 257
JavaScript method: SCFile.getFirst() 258
JavaScript method: SCFile.getLast() 259
JavaScript method: SCFile.getLastRC() 260
JavaScript method: SCFile.getMessages() 261
JavaScript method: SCFile.getNext() 262
JavaScript method: SCFile.getPrev() 264
JavaScript method: SCFile.getPurgedRowCount() 265
JavaScript method: SCFile.getSize() 266
JavaScript method: SCFile.getText() 267
JavaScript method: SCFile.getType() 268
JavaScript method: SCFile.getXML() 270
JavaScript method: SCFile.insertAttachment(attachObj) 271
JavaScript method: SCFile.isRecord() 272
JavaScript method: SCFile.join() 273
JavaScript method: SCFile.JSDate() 274
JavaScript method: SCFile.length() 276

JavaScript method: SCFile.pop() 277
JavaScript method: SCFile.push() 278
JavaScript method: SCFile.setFields() 280
JavaScript method: SCFile.setOrderBy() 282
JavaScript method: SCFile.setType() 284
JavaScript method: scfile_setrecord 285
JavaScript method: SCFile.setValue() 286
JavaScript method: SCFile.shift() 287
JavaScript method: SCFile.toArray() 289
JavaScript method: SCFile.unshift() 290
JavaScript method: SCFile.updateAttachment(attachObj) 291
JavaScript object: SCRecordList 292
JavaScript method: SCRecordList.getCount() 293
JavaScript method: SCRecordList.getPosition() 294
JavaScript object: XML 295
JavaScript method: XML.addAttribute() 297
JavaScript method: XML.addElement() 299
JavaScript method: XML.appendNode() 301
JavaScript method: XML.createNode() 303
JavaScript method: XML.getAttributeNode() 305
JavaScript method: XML.getAttributeValue() 307
JavaScript method: XML.getDocumentElement() 310
JavaScript method: XML.getFirstAttribute() 311
JavaScript method: XML.getFirstChildElement() 314
JavaScript method: XML.getName() 318
JavaScript method: XML.getNextAttribute() 319
JavaScript method: XML.getNextSiblingElement() 323
JavaScript method: XML.getNodeName() 326
JavaScript method: XML.getNodeType() 328
JavaScript method: XML.getNodeValue() 329
JavaScript method: XML.getParentNode() 331
JavaScript method: XML.getPrefix() 335

JavaScript method: XML.getQualifiedName() 338
JavaScript method: XML.getText() 340
JavaScript method: XML.getValue() 343
JavaScript method: XML.importNode() 345
JavaScript method: XML.isDocumentElement() 347
JavaScript method: XML.setAttributeValue() 350
JavaScript method: XML.setContent() 353
JavaScript method: XML.setNodeValue() 355
JavaScript method: XML.setText() 357
JavaScript method: XML.setValue() 359
JavaScript method: XML.toXMLString() 361
JavaScript method: xmlDoc.insertBefore() 362
JavaScript object: XMLDate 363
JavaScript method: XMLDate.addDuration() 366
JavaScript method: XMLDate.getDate() 366
JavaScript method: XMLDate.getDatum() 367
JavaScript method: XMLDate.getGMTSCDateTimeString() 367
JavaScript method: XMLDate.getISODate() 368
JavaScript method: XMLDate.getISODateTimeString() 368
JavaScript method: XMLDate.getISODay() 369
JavaScript method: XMLDate.getISOMonth() 370
JavaScript method: XMLDate.getISOTime() 370
JavaScript method: XMLDate.getISOYear() 371
JavaScript method: XMLDate.getSCDateTimeString() 371
JavaScript method: XMLDate.JSDate() 372
JavaScript method: XMLDate.toSCDuration() 372
List: JavaScript functions 373
JavaScript function: add_graphnodes 374
JavaScript function: getBinary 374
JavaScript function: get_graph_action 374
JavaScript function: get_graph_id 374
JavaScript function: get_graph_node_id 375

JavaScript function: get_graph_target 375
List: JavaScript functions in the ScriptLibrary 375
JavaScript function: addToGroup 376
JavaScript function: checkAllowedOption 380
JavaScript function: checkVersionString 382
JavaScript function: isInGroup 385
JavaScript function: removeFromGroup 387
JavaScript function: returnGroupMembers 392
JavaScript function: returnMyGroups 393
List: JavaScript examples 395
Example: Getting a list of logged-on users 397
Example: Iterating through a complex XML document 397
Example: Query a record using the SCFile object 398
Example: Testing a file name 399
Example: Updating a field with the current date and time 400
Example: Working with structured arrays 401

Chapter 1

Overview

This document is intended for Service Manager system administrators and implementers. It assumes that the reader is familiar with object-oriented programming, the Service Manager System language (RAD), and JavaScript. It provides a brief introduction to RAD, and it provides an overview of JavaScript functionality in Service Manager, using both examples and descriptions of JavaScript's objects, properties, methods, and functions.

System administrators and implementers can use the standard JavaScript scripting language and RAD to customize their Service Manager applications. RAD is the Service Manager system language that enables administrators and implementers to manipulate Service Manager processes and workflows. JavaScript is a widely supported, industry-standard programming language that eliminates the need for a separate programming skill set, and reduces user reliance on proprietary tools. JavaScript in Service Manager does not invalidate the current Service Manager RAD tool set; instead, the JavaScript interface is an alternative to the RAD language for tailoring.The main components of this document are:

- System language (RAD)
- JavaScript and Service Manager
- Where to use JavaScript in Service Manager
- System language reference that describes the RAD functions, rtecalls, and RAD routines
- JavaScript and Service Manager reference that describes the JavaScript global system objects, JavaScript global methods, Service Manager defined JavaScript objects, JavaScript functions, JavaScript functions in the ScriptLibrary, and JavaScript examples

Chapter 2

System language

The System language (RAD) provides the ability for administrators to manipulate the Service Manager processes and workflows using a programming language with a syntax that is similar to the C programming language. Administrators and implementers use this language daily to apply Service Manager to a specific purpose or enterprise. The key to Service Manager's remarkable flexibility can be found in the proper use of the system language.

The System language creates RAD applications that contain application panels (records) linked together in a logical flow. Each panel performs a specific function. When the function is complete, the application exits to another panel. The field value within the exit is a label name of another panel.

The parameter panel defines local variables passed to it by a calling application. In general, two kinds of local variables exist: those used within an application, and those used as exits. Normally, the exit variables are $normal (in the normal exit) and $error (in the error exit).

The first command panel, where execution begins, is always labeled start. Execution continues at a panel's normal exit (or another exit depending on conditions). When that exit is defined as $normal, execution of the panel is complete. If an error is encountered, then the $error exit is taken.

Note: You must have a RAD license to view and modify the RAD applications with the RAD editor.

Data types

The Service Manager data type specifies the expected format of a field's data. The data type determines the possible range of values of the set, the operations that can be performed on the values, and the way values are stored in memory. Defining a data type enables the system to manipulate the data appropriately. The values are denoted either by literals or variables. They also can be obtained as a result of operations. Data types can be either primitive or compound.

Note: System Administrators and implementers should only use types 1 through 4 or 8 to 10.

List: Data types

Data type	Numeric representation	Compound / Primitive	Purpose
number	1	Primitive	Used for all numeric type data.
character	2	Primitive	Used for character strings.
time (date/time)	3	Primitive	Used for absolute dates and times as well as relative time

			intervals.
Boolean (logical)	4	Primitive	Used to represent true or false.
label	5	Primitive	Used to define exits.
File/record	6	Compound	used to define a table or record.
Offset	7	Compound	This is the compiled version of a label. It is used within the code for the go to statements.
Array	8	Compound	Used to represent multiple fields of the same data type.
Structure	9	Compound	Used to represent a collection of fields of the same or different data types.
Operator	10	Primitive	Used in expressions.
Expression	11	Primitive	Used to represent parsed Rapid Application Development (RAD) language expressions.
Pseudo field	12	Compound	Used similarly to functions, it converts input data into output data.
Global variable	13	Compound	Used for variables visible to the whole system.
Local variable	14	Compound	Used for variables only visible to the RAD application

Data type: Primitive

Primitive data types consist of one element. Service Manager internally represents each data type by its numeric representation. Certain functions (`val()` and `type()`) allow the user to determine or modify the numeric representation of a data type, thus changing the data type.

Data type: Number

The number data type is a primitive data type represented by the number 1 in the database dictionary. A number indicates an amount and can be either a whole number, a positive or negative number, or a floating point number. Enter numbers with an optional sign (+ or -) followed by digits which are optionally followed by a decimal point, then more digits which are optionally followed by the letter E and an exponent.
The following number formats display valid numbers accepted by RAD.

Description	Valid literal examples
Integer	1
Negative Integer	-1
Fixed Point	32.1
Floating Point	1.257E05

Furthermore, each bullet shows different representations of the same number:

- 1 = 1.0 = 1E0 = 1.0E0
- -1 = -1.0 = -1E0 = -1.0E0
- 32.1 = 32.10 = 3.21E1
- -32.1 = -32.10 = -3.21E1
- 1.257E05 = 125700 = 1.25700E5

Data type: Character

The character data type is a primitive data type represented by the number 2 in the database. A character data type is typically expressed as a character string. A character string:

- must be enclosed in double quotation marks.
- can be any combination of letters, digits spaces, or special characters.
- can contain any special characters including those used as operators.
- can contain a double quotation mark, but it must be preceded by a backslash (\) to enable the system to distinguish between double quotation marks as part of the string and a double quote at the beginning or end of the string. For example,
  "Vendor=\"473\"" denotes the string: Vendor="437"
- can contain non-characters, such as two hexadecimal digits preceded by a backslash.
- can treat any special or reserved character as a literal character by preceding it with a backslash.
- can present a literal backslash (\) by containing a double backslash (\\)

- can be any length, including zero.
- an empty (for example, length of zero) character string is not the same as a character string with a NULL value.

The following table presents examples:

Description	Literal examples
Alphabetic	"abcde"
Numeric	"12379"
Alphanumeric	"a1276b45"
Special	"$%=+*#@/"
Mixed	"$vendor"
Mixed	"$12379@6:54"
Mixed	"A1276%#B"
Hexadecimal	"\01\ff\c2"
Containing backslash	"C:\\Program Files\\HP\\ Service Manager 7.00\\Server\\RUN\\sm.ini"
Containing double quote	"Enter the \"uid\" in the Unique ID field"

Data type: Time

The Time data type is a primitive data type represented by the number 3 in the database dictionary. In all of the examples in this topic, the following acronyms apply: MM=month; DD=day; DDD=day; YY=year; HH=hours; MM=minutes; SS=seconds. Furthermore, MM/DD/YY is collectively referred to as date and HH:MM:SS is collectively referred to as time. The following rules govern the use of the time data type:

- Time must be enclosed in single quotation marks.
- Time is recorded on a 24-hour clock. `00:00:00' is midnight, `12:00:00' is noon, `18:00:00' is 6 p.m.
- Enter absolute time in the following format: `MM/DD/YY HH:MM:SS'. The format's sequence is affected by the set.timezone() function. It determines the order of date (MM/DD/YY). Note that absolute time is stored as GMT notation.
- Enter relative time in the following format: `DDD HH:MM:SS'
- The use of seconds is optional. If you do not use seconds, the default is :00.
- The use of time is optional in absolute time. If you omit time, it defaults to 00:00:00. For example, 'MM/DD/YY' is `MM/DD/YY 00:00:00'.

- The use of days is optional in relative time. If you omit days, the default is 0. For example '12:00' is zero days and 12 hours, '345 03:00' is 345 days and 3 hours.
- Use the following to set a time variable to a 0 (zero) value: $time = `00:00'

The following date/time formats show valid times/dates accepted by RAD:

Date/time format	Description	Valid literal examples
Absolute time	Indicates a specific date and time	`07/16/83 04:30:00'
Relative time	Indicates an amount of time	`197 05:00' (means 197 days and 5 hours)
Relative time	Indicates an amount of time	'1 00:00:00' (means 1 day)

Data type: Boolean

The Boolean or logical data type is a primitive data type represented by the number 4 in the database dictionary. A Boolean indicates true, false or unknown (of undetermined truth value). The rules for forming a Boolean are as follows:

- Enter the Boolean directly as true, false, NULL, unknown (t, f or u).
- Enter the Boolean in either upper or lowercase characters.

The following formats are accepted.

Description	Valid literal examples
True	t = T= true = TRUE = y = Y = yes = YES
False	f = F = false = FALSE = n = N = no = NO
NULL	Null = NULL
Unknown	u = U = unknown = UNKNOWN

Data type: Label

The label data type is a primitive data type represented by the number 5 in the database dictionary. Labels are used to define exits. The following examples are valid labels.

- $exit
- $normal
- <panel name>

Data type: Compound

Compound data types consist of more than one element. You may combine primitive data types to form compound data types. Any compound data type can have elements of any other valid data types.

Data type: File/Record

The File/Record data type is a compound data type represented by the number 6 in the database dictionary. A file is a set of related records treated as a single unit and is stored on disk.

- A file is a particular kind of structure
- A structure is not a kind of file

The term file is synonymous with the terms relation or table and is sometimes used interchangeably with the term table because of the legacy file system (P4) in the product (ServiceCenter / Service Manager).

Every table must have a table name and other descriptive data in the database dictionary. For example, the name of the contacts table is contacts. The contacts table has keys, fields with index numbers, and a descriptor. Table variables are the mechanism you use to make changes to tables. Always create table variables on the rinit command panel. The table variable stores one record at any given point in time. The record resides in memory for display or modification. For additional information, see the Database Management section in the Service Manager online help documentation.

File variables

A file variable is an instance of a single record in a file. It is not an array, but it is conceptually similar. A file variable contains the dbdict name, the data, and the structure of an entire record as it would be stored in the database according to the dbdict. A file variable has four parts:

- **File name in the database dictionary** The RAD Database Manager requires this to be a single word without spaces or periods. The rinit command panel binds the file variable to a file. The file name of a file variable can be accessed either by using the pseudo field or using the RAD filename() function.
- **List: records selected from the file** When executed, a select command panel retrieves the file's records from the database and sets up the list. When executed, a fdisp (File Display) command panel displays summary information from records in the list.
- **Current record** A file variable can only contain one record from a file at a time. You can move to the next record in the list of records by using the next (Read Next Record) command panel. You can move to the previous record in the list by using the previous (Read Previous Record) command panel. The select command panel always sets the current record to the first record that satisfies the selection query. The fdisp (File Display) command panel sets the current record to the record selected by the cursor when the user presses enter.

- **Descriptor** The descriptor enables fields to be extracted by name. Each record is a structure and has the structure delimiter {[ ]}. The field names in the structure are listed in the database dictionary.

For example:

- $file is a file variable
- lastname is a field name listed in the database dictionary for the file that $file has been bound to using the rinit command panel.
- lastname in $file is a valid expression whose value is the lastname field in the current record in $file, or the nth field, based on the index of the lastname field in the descriptor.

Note: Service Manager is case sensitive. The case for field names, file names, and other elements must match. For example, STATUS.CUST is not the same as status.cust.

Data type: Offset

The Offset data type is a legacy compound data type represented by the number 7 in the Database Dictionary. Offset fields are not supported by RDBMS tables.

Data type: Array

The array data type is a compound data type represented by the number 8 in the database dictionary. Arrays store a list of elements of the same data type accessed by an index (element) number. The term array is synonymous with the terms list, vector, and sequence. Elements in arrays can be of any data types (including arrays or structures). A fully qualified array name (array field in $file) can be used in place of an array variable. The number of items in an array can vary and does not have to be allocated in advance. Arrays are delimited by curly braces ({ }). The following table shows examples.

Description	Literal examples
Numeric	{1,2,3}
Character	{"a","b","c","f","e","h"}
Boolean	{true, true, unknown}
Time	{'12/7/42 00:00', '1/3/62 00:00'}
Nested	{{1,2},{3,4}}
Structures	{{[1,"a"]},{[2,"b"]}} {{[1,"a",true]},{[2,"b",false]},{[3,"c",unknown]}}
Empty	{}

You can use either of the following equivalent syntaxes to access an element in an array:

- $array[element_number]
- element_number in $array

For example, to extract the value of the first customer number (2753) in the array $customer with value {2753, 2842, 2963}, use any of the following equivalent syntaxes:

- 1 in $customer
- $customer[1]

If the accessed array element does not exist, the element is created and set to NULL. This effectively extends the array. To insert a value into an array use the insert RAD function. To delete an element from an array, use the delete RAD function.

Tip: You can use the denull RAD function to remove any null values from the end of an array before assigning the array to a record or adding it to the database.

Data type: Structure

The Structure data type is a compound data type represented by the number 9 in the database dictionary. A structure is a group of named elements of (possibly) differing data types accessed by an index (element) number. Structures are delimited with both curly braces and square brackets ({[ ]}). Note the following examples:

- {[1,"a",true]}
- {[true,‘10/16/90 00:00’,false]}
- {[1,1,"b",0]}

You can use either of the following syntaxes to extract an element from a structure:

- $structure[index of field]
- index of field in $structure

For example, to extract the value of the part number (672) in the part.no field within the structure $order.line1 with the value of {[672,10,"ball.bearings"]} with the field names part.no, quantity and description, use either of the following syntaxes:

- $order.line1[1]
- 1 in $order.line1

In addition, elements of structures may be extracted using their numeric order in the structure . For example, the 1 in $order.line1 is the same as part.no in $order.line1.

Field names are contained in the database dictionary and can only be used when associated with a file variable.

Assuming that the table contains a structure called part and within the structure and there is field called part.no, this command works because it is associated with a file variable.

```
$x=part,part.no in $file
```

These commands do not work because the second command is not associated with a file variable.

```
$y=part in $file
```

```
$x=part.no in $y
```

Data type: Operator

The Operator data type is any expression that is parsed and returns a value, such as tod(), gui(), rtecall(),= (comparison). An operator is a special symbol or function commonly used in expressions. Service Manager uses several different operators:

- arithmetic operators
- logical operators
- relational operators
- special operators
- string operators

The following table presents a list of operators used in Service Manager.

Type	Name	Character	Description
Arithmetic	Addition	+	Indicates that two numbers are to be added together Example: 49 + 51 = 100
Arithmetic	Subtraction	-	Indicates that one number is to be subtracted from another number. To distinguish subtraction from a minus sign, the subtraction operator must be followed by a space. Example: 40 - 20 = 20
Arithmetic	Multiplication	*	Indicates that one number is to be multiplied by another number Example: 5 * 5 = 25
Arithmetic	Division	/	Indicates that one number is to be divided by another number Example: 300 / 10 = 30
Arithmetic	Exponentiation	**	Indicates that the exponential value of a number is to be calculated Example: 2 **5 = 32
Arithmetic	Modulus	mod or %	The modulus is the remainder of a division operation. You may specifically want the remainder for a division operation, or you may want to generate a circular number sequence within a given range. Example:

			5 mod 2 = 1 or 5 % 2 = 1
String	Concatenate	+	Indicates that two strings or two arrays are to be combined (concatenated) into a single string. Example: "a" + "b" = "ab" {"a", "b", "c",} + {"d", "e"} = {"a", "b", "c", "d", "e"}
Logical	not	!	Inverts the Boolean value of the Boolean expression. If the expression is true, the system returns FALSE. If the expression is false, the system returns TRUE. Example: not TRUE = FALSE ~ FALSE = TRUE (UNIX only) UNKNOWN = UNKNOWN ~ UNKNOWN = UNKNOWN
Logical	and	AND and &	Evaluates two expressions and returns a value of TRUE if both expressions are true. If one or both of the expressions is false, the system returns FALSE. Example: TRUE and TRUE = TRUE TRUE and FALSE = FALSE TRUE and UNKNOWN = UNKNOWN FALSE and UNKNOWN = FALSE
Logical	or	OR or |	Evaluates two expressions and returns a TRUE if either or both of the expressions is true. If both expressions are false, it returns a FALSE. Example: TRUE or FALSE = TRUE FALSE | FALSE = FALSE FALSE OR UNKNOWN = UNKNOWN TRUE OR UNKNOWN = TRUE
Relational	Less Than	<	Indicates that the value of one item is less than the value of another item. Example: 400 < 500 is TRUE
Relational	Less Than or Equal To	<= or =<	Indicates that the value of one item is less than or equal to the value of another item. Example: 400 < = 500 is TRUE 400 = < 500 is TRUE
Relational	Equal To	=	Indicates that the value of one item is equal to the value of another item. Example: 1 = 1 is TRUE
Relational	Greater than	>	Indicates that the value of one item is greater than

			the value of another item. Example: ‘08/01/83 00:00’ > ‘07/20/83 00:00’ is TRUE
Relational	Greater Than or Equal To	>= or =>	Indicates that the value of one item is greater than or equal to the value of another item. Example: 600> =300 is TRUE 600= >300 is TRUE
Relational	Not Equal To	(¬ =) or (~=) or (<>) or (><)	Indicates that the value of one item is not equal to the value of another item. Example: 1 ~=2 is TRUE (UNIX only)
Relational	Starts With (Truncated Equals)	#	Indicates that the value of the first string starts with the value of the second string. The order of the operands affects this operation. Example "abc"#"ab" is TRUE
Relational	Does Not Start With (Truncated Not Equal To)	¬# or ~#	Indicates that the value of the first string does not start with the value of the second string. The order of the operands affects this operation. Example "ab" ~ #"abc" is TRUE (UNIX only)
Special	Statement Separation	;	This operator separates two or more statements on the same line Example $A=$B+$C;$B=$C+$D

Special	Parentheses	()	This operator groups together expressions or statements. Service Manager follows the standard order of operations: operators inside the parentheses are evaluated first. Parentheses themselves are evaluated from left to right.
			Example
			3*($x + $y)
			IF ($x=1) THEN ($y="z") ELSE ($y=3)
			IF ($x=1) THEN ($x=2;$z=1) ELSE ($y=3)

Arithmetic operators

An arithmetic operator indicates actions to be performed under the terms of an arithmetic expression. Arithmetic operators have the following precedence: exponentiation, followed by multiplication, division and modulus (all equal); then addition and subtraction (both equal).

Operators with higher precedence are evaluated first. When the operators have equal precedence, they execute from left to right.

Logical operators

A logical operator evaluates one or two Boolean expressions. It determines whether the expression is true, false, or unknown.

The logical operators are as follows (executed in the order shown):

- not
- and
- or

The unknown truth value is treated according to the Substitution Principle, which dictates:

- If UNKNOWN occurs as a logical operand, then the result of the operation is TRUE if substituting TRUE or FALSE for UNKNOWN always yields TRUE.
- The result is FALSE if substituting TRUE or FALSE for UNKNOWN always yields FALSE.
- The result is UNKNOWN if substituting TRUE or FALSE for UNKNOWN sometimes yields TRUE and sometimes yields FALSE.

Relational operators

A relational operator makes a comparison, then generates logical results on whether the comparison is true or false. An operand can be checked for null using the special value NULL; $x=NULL is true if $x is null. Relational operators treat null operands according to the NULL substitution principle:

- If the relation is true regardless of the value of the null operand, the result is true.
- If the relation is false, the result is false.
- Otherwise, the result is unknown.

Data Type: Expression

Expressions are sequences of operators and operands used to compute a value. An expression contains literals or variables. An expression can also be used with a function call. Here are examples of expressions:

- $X * 25
- $Y > 32

Data type: Pseudo field

The pseudo field data type is a legacy compound data type represented by the number 12 in the database dictionary. A pseudo field is similar to a function, because it converts input data to output data and can sometimes be assigned. It can be placed on the left-hand side of the equals (=) sign in an assignment statement. Service Manager has pre-defined pseudo-fields (Month and Name). Pseudo fields were supported for legacy HP ServiceCenter data but are not supported by SQL tables. The following tables describe the pseudo fields.

Month	
Explanation	Accesses the number of months from the beginning of the time base in a date. Use it to increment a date by a period of months.
Format	Month in $time +=1 This increments the current month by one.
Factors	The month pseudo field is useful for scheduling monthly reports, since it calculates the number of days to add to each month.

Name	
Explanation	Accesses the file name in a specific file variable.
Format	$filename=Name in $file This sets $filename to the name of the file bound to $file.
Factors	The name pseudo field is analogous to the filename function

Data type: Global variable

The global variable data type is a compound data type represented by the number 13 in the database dictionary. Global variables begin with `$G.` or `$lo.` and persist throughout a user session (`$lo.` is an abbreviation for logged on. These variables are set when the operator logs on). The server automatically cleans up global variables when a user logs off the system.

Data type: Local variable

The local variable data type is a compound data type represented by the number 14 in the database dictionary. Local variables begin with `$L.` and persist only within the currently executing RAD application. The server automatically cleans up local variables when it exits a RAD application.

Reserved words

Reserved words have special meaning in Service Manager. This meaning is defined in RAD. They can only be used for that purpose. Do not use them for any other purpose such as field names. Here is a alphabetical list of reserved words:

- AND
- BEGIN
- DO
- ELSE
- END
- FALSE
- FOR
- IF
- IN
- ISIN
- NOT
- NULL
- OR
- STEP
- WHILE
- THEN
- TRUE
- UNKNOWN

Literals

A literal is an explicit value. It is expressed as itself rather than as a variable's value or as the result of an expression. Some simple examples are “23”, “2.5”, and “John.” You can use a literal wherever you can use a variable, with one exception. A literal cannot be on the left side of an assignment statement.

Variables

A variable is a location in memory where a value can be stored. It is a named entity that refers to data to which you can assign values. The data type and value of a variable can be different at

different times and can have a primitive or compound data type as its value. Service Manager contains three types of variables:

- Local
- Global
- Thread

Local variables begin with `$L.` and persist only within the currently executing RAD application. The server automatically cleans up local variables when it exits a RAD application.

Global variables begin with `$G.` or `$lo.` and persist throughout a user session (`$lo.` is an abbreviation for logged on. These variables are set when the operator logs on). The server automatically cleans up global variables when a user logs off the system.

Thread variables do not have a consistent naming scheme other than beginning with a dollar sign ($). Thread variables are only valid for the current RAD thread. If the RAD thread terminates, then the server automatically cleans up all thread variables.

Creating variables

The rules for creating variables are as follows:

- A variable name must begin with a dollar sign ($) and an alphabetic character followed by zero or more alphanumeric characters that may include periods, but may not include blanks.
- A variable name can be any length.
- A variable's value is initialized to a null string.
- Data assigned to a variable has a data type. This data type becomes the variable's data type.
- Any variable can be set to a null value by assigning the value NULL into it. For example, $variable=NULL.
- Global variables are treated as global among all applications executed within the process or task with two exceptions:
  a. When the variable is designated local (or something other than global) by an initial identifier. For example, when the variable is created within a function.
  b. When the variable is a parameter variable.

Global variables

The following global variables are available when defining registration events.

Variable	Description	Comment
$axces	Represents the eventin record.	
$axces.fields	Represents the evlist field in the eventin record.	
$axces.register	Represents the event registration record.	

$axces.lock.interval	An interval of time before a retry occurs if HP Service Manager denies the attempt to update an incident because the record is locked. For example: '00:02:00' for two minutes.	
$axces.debug	If true, the evlist array in the eventin record is not removed before attempting to update the record. If the size of the record exceeds 32KB, Event Services generates an error, the eventin record is not updated, and the event reprocesses because the evtime field is not removed.	Use this feature with discretion.
$axces.bypass.failed.validation	Used in events calling the application axces.apm. The default value is NOT true. If true, the application ignores any failed formatctrl validations. If false, the event status is error-fc.	

List: Default variables

It is likely that the list of default variables varies from release to release. The out-of-box system includes the following global and local variables.

Variable	Definition
$G.profiles	List of personal profiles.
$G.pm.global.environment	Incident Management's global environment record .
$G.assignment	List of assignment groups for Service Desk, Incident Management, Problem Management and Change Management.
$G.categories	List of categories.
$G.problem.inboxes	List of Incident Management views available to all the users as well as the views available to the logged on operator.
$G.open.lists	List of all global lists built at login.
$G.icm.status	List of Configuration Management statuses.
$G.assignment.groups	List of assignment groups for Service Desk, Incident Management, Problem Management and Change Management.
$G.prompt.for.save	Global flag (Boolean) that determines whether or not to prompt you to save when you press OK or Cancel after you update a record.
$G.availability.maps	List of availability maps.

$G.pm.environment	The current user's Incident Management personal profile record.
$G.pm.status	List of Incident Management statuses.
$G.technicians	List of technicians (operator table).
$G.incident.inboxes	List of Service Desk views available to all the users as well as the views available to the logged on operator.
$L.filed	This is the current file opened by the display application.
$L.new	The current file being updated when evaluating macro conditions.
$L.old	The pre-updated state of a file undergoing an update while evaluating macro conditions.
$lo.appl.name	The name of the application responsible for managing the user's menu (menu.manager)
$lo.appl.names	Application parameter names passed.
$lo.appl.values	Application parameter values passed.
$lo.cm.limit	Partial-key time threshold for Change Management.
$lo.company.name	The company name as defined in the System Information Record as Menu Title under the Menu Information tab.
$lo.copyright	HP copyright (copyright())
$lo.date.order	The number representing date order (mdy, dmy, ymd) in the operator record.
$lo.db.limit	Partial-key time threshold for Database Manager.
$lo.device	Device ID (current.device())
$lo.groups	List of query group names in operator record.
$lo.home	Name of Home menu for GUI.
$lo.i	Temporary variable that functions as a loop counter (variable is cleaned up in the login application).
$lo.main	Name of main menu for text.
$lo.month.abv	List of abbreviated months ("Jan", "Feb", "Mar").
$lo.month.ext	List of months ("January", "February", "March").
$lo.msglog.lvl	Level of messages to store in the message queue for the operator Level of messages to store in the message queue for the operator logged on. Message levels affect on-screen messages only.
$lo.pm.limit	Partial-key time threshold for Incident Management .
$lo.system.startup	Set at login to value of the field called system.startup.time in

	Company Information record.
$lo.time.zone	User's time zone.
$lo.uallow.syslog	Determines whether syslog table is updated when processes start and end in the system.
$lo.uallow.timezone	Allows user to modify time zone based on capability.
$lo.uapprovals	List of change approval groups.
$lo.ucapex	List of user's capability words.
$lo.uchgrps	List of change groups.
$lo.ufname	User's full name.
$lo.ulogin.time	User login time stamp.
$lo.user.name	operator()

Variable pools

Variable pools are functional groupings of variables within Service Manager. There are currently four variable pools:

- **Global**
  A global variable is visible to the entire system. It begins with $G. or $lo..
- **Local**
  A local variable is only visible to the RAD application in which it was defined. It begins with $L..
- **Parameter**
  A parameter variable is defined on a parameter command panel. It may contain a value passed in from another application. By convention, parameter variables are written in uppercase letters, such as $PHASE or $GROUP.LIST. Parameter variables are invisible to the debugger.
- **Thread**
  A thread variable is only visible to the thread in which it was defined. The same variable in different threads has different values, even when the threads are spawned by the same parent.

Variable pool	Variable	Comments
Global	$G.	
Global	$lo.	
Global	$MARQUEE.	
Global	$SYSPUB.	
Local	$L.	There is only one local variable. It can contain many pieces of

		data, similar to the way a structure contains many pieces of information. For example, $L.env and $L.file are two pieces of data that $L. can contain.
Parameter	$PHASE $GROUP.LIST	These are just a few examples, there are many parameter variables.
Thread	$file $array $post	These are just a few examples, there are many thread variables.

Statements

Statements are the smallest executable entities within Service Manager. A statement is governed by these rules.

- A statement does not have a value.
- A statement is comprised of expressions combined with key words.
- The following BASIC language statements perform processing and looping: IF, WHILE and FOR. They can be nested. For example, for...to...if...then...else...if...then.
- The following assignment statements are used: assign, increment, and decrement.
- RAD functions can be used in statements to further define selection criteria and/or execute commands, initialize values, and perform calculations. Service Manager syntax rules must be followed. The result of a RAD function can be assigned to a variable using the assign (=) assignment statement. For example, `$string.length=lng($string)`.

The following table lists all of the statements available in Service Manager.

Type	Name	Character	Description and Examples
Assignment	Assign	=	Assigns the value of the right hand operand to the left hand operand. Example: $x=1 assigns 1 to $x
Assignment	Increment	+ =	Increment the left hand operand by the right hand operand. Example: traditional: $x=$x+1 increments $x by 1 shortcut: $x += 1 increments $x by 1
Assignment	Decrement	- =	Decrement the left hand operand by the right hand operand. Example: traditional: $x=$x-1 decrements $x by 1 shortcut: $x -= 1 decrements $x by 1

Processing and looping	FOR		Allows you to perform a loop. You can set a variable, perform a statement, and increment the variable until the variable is greater than a maximum value. Format this statement as follows: `FOR variable name = initial value TO maximum value [DO] statement` The brackets ([ ]) indicate that the DO keyword is optional. Example: for $I = 1 to 10 do ($J=$I*$I+$J)

Processing and looping	IF		Specifies a condition to be tested and a statement to be executed if the condition is satisfied and a statement to be executed if the condition is not satisfied. Format these statements as follows:
			`IF Boolean condition THEN statements [ELSE statements]`
			Example:
			`if ($location="Seattle") then ($x=$x+1) else ($x=$x - 1)`
			If Boolean condition evaluates to UNKNOWN, then neither statement is executed; therefore, HP recommends using a default value when the condition evaluates to UNKNOWN.
			Example:
			`if (nullsub($location, "Chicago")="Seattle") then ($x=$x+1) else ($x=$x - 1)`

Processing and looping	WHILE		Specifies a condition to be tested and a statement to be executed when the condition is TRUE. Format these statements as follows:
			`WHILE (expression) [DO] statement`
			Example:
			`while ($x>6) do ($x=$x - 1;$y=$y - 1)`
			The brackets ([ ]) indicate that the DO keyword is optional.

Assignment statements

An assignment statement assigns a value to a variable. The order for the assignment statements when read left to right is the destination variable first, then the assignment operator, then the expression. It usually consists of three elements:

- an assignment operator (typically an =)
- an expression to be assigned
- a destination variable

On execution, Service Manager evaluates the expression and stores the resulting value in the variable.

List: Command line calls

You can use the Service Manager command line to navigate through the Service Manager system and to run applications, wizards, and macros.

Generic command line calls

Command	RAD application called	Function	Example
***a<application name>**	as specified	Runs the specified RAD routine.	*adatabase accesses Database Manager.
***f<file name>**	database	Accesses a specified file.	*fcontacts accesses the contacts file.
***q<query name>**	query.stored	Runs a stored query.	*qIM.open.ALL displays a list of all open incidents.
***s<script name>**	script.execute	Runs the specified script.	*socmq.open.hr.quote opens an HR Request ticket.
#<shortcut name>	non-applicable	Queries for a list of shortcut commands.	#do presents a list of commands that start with “do.”

Direct command line calls

Command	RAD application called	Function
about	bulletin.key	Displays the system bulletin and any active “hot”

		incidents.
ag	encl.appl	Displays the RAD Editor prompt (formerly known as Application Generator).
agcompare	compare.applications	Displays the RAD Application Comparison form.
agentrev	query.stored	Executes the “agentrev” query which displays a list of the schedule records for all SCAuto agents.
agentstat	scauto.check.setup	Displays a currently-available SCAuto agents. Form includes options for stopping agents
agmap	agmap.sched	Displays the Application Mapping form.
agr	report.exerciser	Executes the “OPEN PROBLEM ANALYSIS REPORTS - ASSIGNMENT GROUP” report via Report Exerciser.
alert	database	Displays the Alert Definition form.
alertdefs	se.search.engine	Displays the Alert Definition form.
alertlogs	se.search.engine	Displays the Alert Log form.
appr	database	Displays the Approval Definition form.
approvaldefs	se.search.engine	Displays the Approval Definition form.
approvallogs	se.search.engine	Displays the Approval Log form.
asr	report.exerciser	Executes the “ALERT STATUS REPORT” report via Report Exerciser.
auditdelta	audit.unload.front	Displays the form for unloading an Audit Delta.
audithist	database	Displays the Audit History form.
auditonoff	database	Displays the Audit Control form.
audlog	database	Displays the Audit Log form.
audspec	database	Displays the Audit Specification Table form.
autoshut	database	Displays the Automatic Shutdown Information form.
bm	database	Displays the form for accessing Benchmark field information.
bmconfig	database	Displays the Benchmark Configuration Record form.
bmparms	database	Displays the Benchmark Parameters form.
bulletin	database	Displays the System Bulletin form.

calduty	database	Displays the Normal Working Hours form.
calholiday	database	Displays the Holiday Names form.
call	exercise	Displays the Application Exerciser form for testing/running a RAD application directly.
callqueue	sc.setup.manage	Displays the Interaction) queue.
chgqueue	sc.setup.manage	Displays the Change queue.
cls	pm.access	Displays the Incident queue.
cm3grp	database	Displays the Change Management Groups Definition form.
cm3msg	database	Displays Change Management Event Definition form.
cm3profile	database	Displays the Change Management (ChM) Security Profile form.
cm3r	se.search.engine	Displays the search form for Change Management requests.
cm3rcat	cm3r.category.maint	Displays the Change Management Request Category form.
cm3renv	se.search.engine	Displays the Change Management Application Environment form.
cm3ropen	cmc.open.from.menu	Initiates the script for opening a new change request ticket.
cm3t	se.search.engine	Displays the search form for Change Management tasks.
cm3tcat	cm3t.category.maint	Displays the Change Management Task Category form.
cm3tenv	se.search.engine	Displays the Change Management Application Environment - Tasks form.
cm3topen	cmt.open.from.menu	Initiates the script for opening a new change task.
cmcontrol	cm.edit.config	Displays the Contract Management Configuration form.
cmdlist	report.exerciser	Executes the Menu Command List report via Report Exerciser.
comp	se.search.engine	Displays the System Information Definition form (system Company record).
compare	compare.applications	Displays the RAD Application Comparison form.

configure sla	sla.edit.config	Displays the Service Level Agreement Control form.
contacts	database	Displays the Contacts search form.
copyfile	copy.database.file	Initiates the script for copying a database file.
createschd	database	Displays the Schedule form.
curalerts	se.search.engine	Displays the Alert search form.
curapprovals	se.search.engine	Displays the Approval search form.
curconvert	database	Displays the Currency Conversion search form.
currency	database	Displays the Currency form.
datamap	database	Displays the Data Map form.
db	database	Displays the Database Manager form.
dbdict	dbdict.utility database	Displays the Database Dictionary prompt.
de	database	Displays the Display Application Event Definition form.
delmail	database	Displays the form for sending internal Service Manager broadcast mail.
dist	database	Displays the Distribution Group form.
do	database	Displays the Display Application Option Definition form.
doc	menu.manager	Displays the menu with options for maintaining the document engine. **Note**: This command line call will only work if the menu forms are being displayed.
ds	database	Displays the Display Application Screen Definition form.
edit gkn	database	Displays the form for editing core knowledge.
err	report.exerciser	Executes the “EXCESSIVE REASSIGNMENT REPORT” report via Report Exerciser.
evemail	sca.window	Displays the form for sending an email event.
eventbld	axces.build.maps	Initiates the wizard for building event mappings.
eventfltr	database	Displays the Event Filters form.
eventin	database	Displays the Event Services input queue.

eventmap	database	Displays the Event Map form.
eventout	database	Displays the Event Services output queue.
eventreg	database	Displays the Event Registration form.
evfax	sca.window	Displays the form for sending a fax event.
evicmbld	scauto.build.maps	Initiates a script for building Configuration Management event mappings.
evpage	axces.page.window	Displays the form for sending a paging event.
evwrite	sca.window	Displays the form for sending an Incident Management, Configuration Management, or generic event.
expline	database	Displays the Expense Line Information form.
faxinfo	database	Displays the Fax Configuration form.
fc	format.ctrl.maint	Displays the Format Control form.
fd	forms.designer	Displays the Forms Designer form.
files	database	Displays the Configuration File form for database unloads.
formatctrl	format.ctrl.maint	Displays the Format Control form.
forms	forms.designer	Displays the Forms Designer prompt.
gen core	ke.gencore.verify	Displays the prompt for generating the core Knowledge Base.
gl	database	Displays the global list form.
hd	pm.access	Displays the Incident queue.
htopic	database	Displays the Help topic search form.
imapplenv	database	Displays the Incident Management Environment Profile form.
imassign	database	Displays the Assignment Groups form.
imbuild	pm.build.probsum	Initiates the script for the building of probsummary records.
imconvert	pm.convert.format	Displays the Convert Incident Format form.
imgroups	database	Displays the Incident Management Groups form.
improfile	database	Displays the Incident Management Security Profile form.

imreset	pm.clear.downtime	Allows for the clearing of availability information.
inact	kill.inactive.setup	Displays the Kill Inactive Users Setup Parameters form.
incidentqueue	sc.setup.manage	Displays the Incident queue.
info	database	Displays the Background Processor Initialization Registry form.
irassg	report.exerciser	Executes the “Problem Response Analysis By Assignment” report via Report Exerciser.
ircomp	report.exerciser	Executes the “Problem Response Analysis By Component” report via Report Exerciser.
irloc	report.exerciser	Executes the Problem Response Analysis By Location report via Report Exerciser.
irq	ir.query.window	Displays the form for issuing an IR Query File form.
irvend	report.exerciser	Executes the “Problem Response Analysis By Vendor” report via Report Exerciser.
knowledge	get.search.application	Displays the search form for Knowledgebase information.
link	edit.link	Displays the Link File form.
linklist	report.exerciser	Executes the Link File Listing report via Report Exerciser.
load transfer	apm.upgrade.load.transfer	Initiates the Service Manager Upgrade Utility Load Transfer form.
location	database	Displays the Location form.
logmsg	database	Displays the “log” type Message Class form.
mailmsg	database	Displays the “mail” type Message Class form.
mcf	database	Displays the Capability Word form.
menu	database	Displays the Menu form.
message	database	Displays the Message form.
model	database	Displays the Configuration Item (CI) Model search form.
modelicm	database	Displays the Configuration Item (CI) Model search form.
modelven	database	Displays the Model Vendor Information form.

monitor	system.monitor	Displays the system status form.
msg	database	Displays the generic Message Class form.
msglog	database	Displays the Message Log form.
msgtype	database	Displays the Message Type form.
new call	cc.first	Displays a blank form to create a Service Desk interaction.
new incident	apm.first	Displays a blank incident form for ticket entry.
note	database	Displays the Notification Definition form.
number	database	Displays the Sequential Number form.
ocmbldlvl	ocm.co.bld.lvls	Displays the prompt for updating all of the level numbers for all components in the model file.
ocmco	database	Displays the Configuration Item (CI) Model search form.
ocmdeliv	report.exerciser	Executes the "Delivery Forecast for Next 7 Days" report via Report Exerciser.
ocmdreqj	report.exerciser	Executes the "Daily Quote Log" report via Report Exerciser.
ocmdrj	report.exerciser	Executes the "Daily Receipt Journal" report via Report Exerciser.
ocmevents	database	Displays the Request Management Events form.
ocmgroups	database	Displays the Request Management Group Definition form.
ocml	ocml.access	Displays the search form for Request Management line items
ocmlcat	database	Displays the Request Management Line Item Category form.
ocmlcreate	ocm.category.create	Displays the Request Management Line Item Category form.
ocmlenvir	query.stored	Displays the Request Management Line Items Application Environment form.
ocmlmast	database	Displays the Request Management Catalog Select Categories form.
ocmlphase	database	Displays the Request Management Line Item Phase form.
ocmlrec	database	Displays the Request Management Receiving

		Log form.
ocmo	ocmo.access	Displays the search form for Request Management orders.
ocmoappr	ocmo.access	Displays a list of orders that have been approved.
ocmocat	database	Displays the Request Management Order Category form.
ocmocavail	database	Displays the Request Management Check Availability Order Generation Schedule Record.
ocmocreate	ocm.category.create	Displays the Request Management Order Category form.
ocmodemand	database	Displays the Request Management Background Order Generation Schedule Record form.
ocmoenvir	query.stored	Displays the Request Management Orders Application Environment form.
ocmoopen	ocmo.access	Initiates the script for creating a new Request Management order.
ocmophase	database	Displays the Request Management Order Phase form.
ocmpo	report.exerciser	Executes the "Print Purchase Orders" report via Report Exerciser.
ocmprofile	database	Displays the Request Management Profile form.
ocmq	ocmq.access	Displays the search form for Request Management quotes.
ocmqcat	database	Displays the Request Management Quote Category form.
ocmqcreate	ocm.category.create	Displays the Request Management Quote Category form.
ocmqopen	ocmq.access	Initiates the script for creating a new Request Management quote.
ocmqphase	database	Displays the Request Management Quote Phase form.
ocmvenper	report.exerciser	Executes the "Vendor Performance Based on Line Item Dates" report via Report Exerciser.
odr	report.exerciser	Executes the "OPEN PROBLEM ANALYSIS REPORTS - DOCUMENT NUMBER" report via Report Exerciser.

oncall	database	Displays the On Call Schedule contact form.
oncallsched	database	Displays the On Call Schedule calendar form.
operator	database	Displays the Operator form.
opn	pm.access	Displays the Incident queue.
overspent	query.stored	Executes the "overspent contracts" query which displays a list of all service contracts that have exceeded their financial budget.
password	password.change	Displays the prompt for changing the login password.
print	report.exerciser	Executes the "Print of Application" report via Report Exerciser.
printappl	report.exerciser	Executes the "Print of Application" report via Report Exerciser.
prtmsg	database	Displays the "print" type Message Class form.
purgarch	pa.main.appl	Displays the Purge/Archive form.
purgeaudit	audit.purge	Displays the form for purging audit history.
RAD	encl.appl	Displays the RAD Editor prompt.
rad	encl.appl	Displays the RAD Editor prompt.
re	report.exerciser	Displays the Report Exerciser prompt.
recalc partials	sla.recalc.totals	Displays the prompt for recalculating SLA information over a given time frame.
report	report.exerciser	Displays the Report Exerciser prompt.
review cc	cc.setup.manage	Displays the Service Desk Interaction queue.
review im	apm.setup.manage	Displays the Incident queue.
rmail	read mail	Displays the prompt for selecting which type of internal Service Manager mail to read.
run	exercise	Displays the Application Exerciser form for testing/running a RAD application directly.
rv	pm.access	Displays the Incident queue.
rw	report.writer	Displays the Report Writer prompt.
sch	database	Displays the Schedule File form.
schedule	database	Displays the Schedule File form.

schmail	mail.set.schedule	Displays the form for scheduling background mail.
scknowledge	get.search.application	Displays the Knowledgebase search form.
scmsg	database	Displays the Service Manager Message form.
scr	report.exerciser	Executes the "shift change report" report via Report Exerciser.
scripts	script.maint	Displays the Script Definition form.
search cal	cc.search.incidents	Displays the search form for Service Desk interactions.
search calls	cc.search.incidents	Displays the search form for Service Desk interactions.
search incident	apm.search.problems	Displays the search form for Incident Management incidents.
search sla	sla.search.objects	Displays the search form for Service Level Agreements.
servcont	database	Displays the Service Contract form.
shutdown	system.shutdown	Initiates the process for shutting down the Service Manager server.
smail	mail.send.appl	Displays the prompt for confirming the sending of internal Service Manager mail.
smapplenv	database	Displays the Service Desk Environment form.
smgroups	database	Displays the Service Desk Groups form.
smprofile	database	Displays the Service Desk Security Profile form.
software	database	Displays the Software Management form.
sol cans	se.search.engine	Displays the Pending Knowledge Items form.
spool	spool.scheduler	Displays the Spoolheader form.
sq	database	Displays the Stored Query Maintenance form.
start	apm.upgrade.main	Initiates the Upgrade Utility wizard.
startbm	benchmark.start	Displays the Benchmark Utility form.
status	system.status	Displays the system status form.
stdptrs	database	Displays the Configuration File form for standard printers.
subtotals	database	Displays the Subtotal Definition form.

syslog	database	Displays the System Log form.
task	query.stored	Displays a list of the user's change tasks.
terminals	database	Displays the Terminal Configuration form.
triggers	database	Displays the triggers form.
tskqueue	sc.setup.manage	Displays the Task queue.
unload	us.unload	Displays a list of unload scripts.
uoperprof	query.stored	Displays the Operator Status Display form.
upd	pm.access	Displays the Incident queue.
userlevel	us.userlevel	Displays the current user access level and the options for changing this access.
userstats	report.userstats	Displays the prompt for executing the Usage Statistics report.
usrcmr	cm3r.main	Initiates the script for opening a new change request record.
usrdir	database	Displays the Contacts form.
usricm	icm.access	Displays the Configuration Item Search form.
usrknow	database	Displays the search form for Knowledge Base information.
usrocma	ocmo.access	Displays a list of the current Request Management orders.
usrocmo	ocmo.access	Displays the search form for Request Management orders.
validity	se.search.engine	Displays the Validity Table Specifications form.
vendor	database	Displays the Vendor/Manufacturer search form.
viewbm	benchmark.avg.results	Displays the Benchmark Results.

Call a RAD application

You can call a RAD application from the following places in Service Manager:

- Display Options
- Format Control
- Scripts

- Processes
- Triggers

When you call a RAD application, you may need to pass the required parameters. These parameters consist of a parameter name and a value.

Note: Service Manager does not currently support calls from JavaScript on any RAD application that uses the rio/fdisp panels. Avoid calls from JavaScript on RAD applications that use the rio or fdisp panels. If you cannot avoid doing so, always set the RunInNewThread parameter to true. This does not work for all cases.

To find view the parameters:

1. Locate the RAD you want to use. See "Locate a RAD function" below.
2. Click **View** when you have the RAD application panel displayed. This displays a list of parameter to pass.
3. Position the cursor on each parameter and then click Help to view the name of the field used as the parameter.

Note: To view field help, you must have the setting, Show context-sensitive help debug information selected. Windows client: **Preferences > HP Service Manager > Appearance**.

Locate a RAD function

1. Click **Tailoring** > **Database Manager**.
2. Type =application in the Form field.
3. Click **Search**. A blank application file record displays.
4. Click **More** or the **More Actions** icon and then select Expert Search. A query window displays.
5. Type one of the following in the Query field:

Desired search	Syntax

Search for a function that is not an rtecall function.	`index("<function name>",`
	`str(contents(currec())))>0`
	Example:
	`index("jscall", str(contents(currec ())))>0`
	Returns all the RAD application panels that contain the function “jscall” (to call a JavaScript function).

Search for all instances of a particular rtecall function. For this query to run properly, a backslash must precede	`index("rtecall(\"<function name>\"",` `str(contents(currec())))>0`

each double quotation mark within the rtecall parentheses.	Example: `index("rtecall(\"rinit\"", str (contents(currec())))>0` Returns all the RAD application panels that contain the function “rinit” (to initialize a file).

Use this syntax to search for all instances of all rtecall functions beginning with a particular letter (for example, the letter r). For this query to run properly, a backslash must precede the first set of double quotation marks. The closed parenthesis is not necessary in this syntax.	`index("rtecall(\"<first letter>",`
	`str(contents(currec())))>0`
	Example:
	`index("rtecall(\"s", str(contents (currec())))>0`
	Returns all the RAD application panels that contain a function that starts with the letter “s” such as “sort”.

6. Click **Search**. A record list of RAD panels matching the search criteria displays.
7. Double-click on a panel to display that instance of the function.

RAD Debugger

The RAD Debugger enables you to view and isolate RAD processes for the purpose of troubleshooting. Information about the RAD flow opens in a separate window containing a scrollable text field. The RAD Debugger uses its own command line language to trace, display values, and set break points.

Start RAD Debugger

1. Click **Window** > **Show View** > **Other** > **HP Service Manager Administrator** > **RAD Debugger**.
   The RAD Debugger command window opens.
2. Type the desired command followed by a parameter.
3. Press **Enter**.

RAD Debugger commands

The following is a list of the commands used in the RAD Debugger to troubleshoot an application:

Command	Description
d (display)	Displays the contents of a variable. A common use of this command is to display the name of the Display application Screen ID attached to the current form. d $L.screen
t (trace)	Turns tracing on or off. Tracing allows you to see every panel that the RAD flow encounters. Use the command by itself to display the status of the trace function. t on t off t
ta	Turns panel tracing on or off for a specific RAD application. To turn application tracing on, include the name of a RAD application. To turn application tracing off, repeat the same command combination a second time. You may have multiple traces engaged at one time. Use the command by itself to display a list of the RAD applications being traced. ta script.execute ta
tar	Removes all RAD specific panel traces. If you are tracing multiple applications, you can save time by using tar rather than using ta to turn off each trace individually.
b (breakpoint)	Sets a panel breakpoint. When the RAD flow encounters a panel with this label, it halts before executing the named panel and gives you an opportunity to perform debugging procedures such as displaying variables or executing statements. Repeat the command combination a second time to turn off the breakpoint. Execute the command by itself to display a list of all current breakpoints. b init.operator b
ba	Sets a breakpoint on a specific RAD application. A breakpoint is applied each time the RAD flow enters the named application, either when it runs for the first time or when the flow returns to it from a subroutine. ba menu.manager
bt	Sets a breakpoint on a specific type of RAD panel. When the RAD flow encounters a RAD command panel with this name, the application stops running and returns you to the RAD Debugger. Enter the command combined with the name of a RAD panel to set the breakpoint. Re-enter the same command combination to turn off the breakpoint. bt rinit
bv	Set a breakpoint when a variable changes.

rb	Removes all breakpoints of any type.
go	Resume execution after a breakpoint.
c (continue)	Resumes the execution of the RAD flow after a breakpoint occurs.
s (step)	Steps through the RAD flow one panel at a time.
gl (globals)	Displays all the global variables—those beginning with $G., $lo., $CHART., $SYSPUB., $MARQUEE.
v (variables)	Displays all the thread variables: those not beginning with a special code.
l (locals)	Displays all the local RAD variables: those beginning with $L.
sta (stack)	Displays the current RAD stack.
re (relations)	Displays only the variables which are table relations (file handles initialized by the rinit command panel).
m (memory)	Displays all the variables in memory. This is equivalent to issuing the globals (g), variables (v), and locals (l) commands.
x	Execute a statement.
display	Display name - view a variable.
h (help)	Displays a brief summary of the RAD Debugger commands.

Chapter 3

Introduction to JavaScript in Service Manager

Like RAD, JavaScript in Service Manager provides the ability to query, insert, update, and delete database records, and to make use of existing RAD variables and functions. The principal benefit of using JavaScript is that it uses an industry-standard scripting language to execute commands instead of a proprietary language. JavaScript within Service Manager is implemented using the Mozilla® SpiderMonkey (JavaScript–C) Engine.

For detailed information about the syntax of Mozilla SpiderMonkey JavaScript, refer to:

http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide

For information about the SpiderMonkey project refer to:

http://www.mozilla.org/js/spidermonkey/

Language overview

JavaScript is a scripting language which is different from programming languages like Java™, C++, or Visual Basic®. It is a smaller, dynamically typed, scripting language that offers programming tools with easy syntax and built-in functionality.

Unlike programming languages, scripting languages do not use compilers to create executable program code. Because JavaScript is an interpreted language, each time the script is executed, it is loaded into an interpreter that runs the code. The Compile button in the Script Library does not create an executable load module; it runs a syntax check to verify that the script is executable.

JavaScript can be either client-side, server-side, or core language. Core JavaScript is the base JavaScript language with client-side and server-side JavaScript as extensions. Server-side JavaScript is used for accessing and manipulating data and is the only one available within Service Manager.

Learning JavaScript

HP does not endorse these references but provides them as a courtesy to our customers and partners. These are helpful resources that contain information about JavaScript.

- For JavaScript basics, take the JavaScript tutorial(s) at W3 Schools.
- The Mozilla SpiderMonkey (JavaScript-C) Engine project.
- *JavaScript: The Definitive Guide, 5th Edition* by David Flanagan (ISBN: 0596101996).

For best results, fully understand the programming language prior to attempting to use it with HP Service Manager.

Note: There is a difference between server-side and client-side JavaScript. Service Manager only supports server-side JavaScript. Therefore, any JavaScript that addresses output is not available in the Service Manager environment.

Basic rules of JavaScript

The following rules apply to the basic syntax of JavaScript:

- JavaScript is case-sensitive.
- Statements should end in a semicolon (;).
- Variables:
  - Must be defined before being used. The variable name can contain A – Z, a – z, underscore or digits and must start with a letter or an underscore ("_").
  - Assume the type of the data that is put into the variable. The data type does not have to be explicitly defined
  - Are global variables when defined outside of a function, and are available anywhere in the current script context. Variables created within a function are local variables, and can be used only within that function.
- Strings have to be enclosed in quotation marks, either a single or double. For example: print ("Hello " + 'world '+ Country.name) produces the following: Hello world US.
- Special characters that are displayed literally must be preceded by a backslash character (\). Quotes within a string can be entered preceded by a backslash as well. See the Core Language Features - Literals - "String literals" section for more information in http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide.
- To increment a variable, such as a = a + 1, you can use a++. You can decrement a variable in the same way, as in a--.
- To enter comments in the script, use "//" to start a single line comment or the combination of "/*" and "*/" to enclose a multi-line comment.
- Values that are not defined as a data type (string, number, Boolean) may be defined as an object, such as Date, Array, Boolean, String, and Number. As an example you could define: var ArrayList=new Array("test", " this", " list");
- Dots in Service Manager field names must be replaced by an underscore (_) in JavaScript. For example contact.name becomes contact_name.
- Service Manager field names that are reserved words in JavaScript have to be preceded by an underscore, such as "_class" for the "class" field.

Structure of a JavaScript application

```
<Global variable declarations;>

function <function name>(<parameters>)

{

        <local variable declarations;>
```

```
        <Statements;>
}
<function calls;>
```

Expressions and operators

For more information on this topic refer to the Expressions and Operators section in:

http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide

The list of operators includes assignment operators, comparison operators, arithmetic operators, bitwise operators, logical operators, string operators, and special operators.

JavaScript objects and properties

The key to understanding JavaScript is learning the structure of its objects. Most JavaScript functionality is contained in objects. The objects contain methods, parameters, and events. Each object may contain a set of properties that more closely define the object. The syntax you use to call methods and properties is: Object.method or Object.property.

For more information about JavaScript objects and properties refer to the Details of the Object Model section in :

http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide

Built-in objects and functions

Built-in objects described in this section are Array, Boolean, Date, Function, Math, Number, and String.

Each of these is described in detail in the Working With Objects – Predefined Core Objects section of:

http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide

Conditional statements

The following example returns true if the date entered is within the number of days in a month (here, 30 days); otherwise, it returns false.

Note the use of the conditional statement to shorten the "if" expressions. The statement:

```
var return = (parm.Date.getDate <= 30) ? true : false;
```

works in the same way as its much longer equivalent:

```
var result=false;
if (parm.getDate()<= 30)
{
   result = true;
}
else
```

```
{
   result=false;
}
```

While and loop statements

Note: Whenever programming a loop statement, ensure that the exit condition will be met; otherwise an infinite loop will occur.

For detailed information about the for and while loop statements, refer to Statements – Loop Statements section in:

http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide

The following example shows how to check how many times a given number can be divided by 2, first using a "for" loop, and then using a "while" loop:

```
function DivisableByTwoFOR(ParmNumber)
{
       for (var i=0; ParmNumber > 1; i++)
       {
               ParmNumber = ParmNumber / 2;
       }
return i;
}
function DivisableByTwoWHILE(ParmNumber)
{
       var i = 0;
       while( ParmNumber > 1 )
       {
               ParmNumber = ParmNumber / 2;
               i++;
       }
return i;
}
```

A Do..While loop is available for those instances when you want to execute a statement (or block of statements) at least once. In a Do..While loop the exit condition is checked after the statements are executed.

When enhancing an array object with a custom method, the for..in-loop iterates on all items in the array including the custom method. This might lead to errors that are difficult to debug. You can use a guarding `if` statement to help debug the code.

For example:

```
if (<array> .hasOwnProperty(<item index>))
```

or

```
if (typeof(<array>[<item index>]!="function")
```

Break and continue statements

The break statement terminates a "while" or "for" loop completely. The continue statement terminates execution of the statements within a "while" or "for" loop and continues the loop in the next iteration. The following two examples demonstrate how these statements are used.

Note: Both these statements violate the rules of structured programming but are nonetheless widely used in special cases.

```
function break4error(parmArray, x)
{
    var i = 0;
    var n = 0;
    while (i < 10)
    {
       if (isNaN(parmArray[x]))
    {
       print(parmArray[i].toString + " is not numeric. Please repair
       and run again");
       break;
    }
       n = n + parmArray[i];
       i++;
    }
    return n;
}
```

Modifying the above example, the continue statement would work as follows:

```
function break4error(parmArray, x)
{
    var i = 0;
    var n = 0;
    while (i < 10)
    {
       if (isNaN(parmArray[x]))
       {
           print(parmArray[i].toString + " is not numeric. Continuing with
           next number");
           i++;
           continue;
       }
           n = n + parmArray[i];
           i++;
       }
       return n;
}
```

Exception handling statements

As a best practice, exception handling should always be included in any JavaScript program. The following code provides an example of using exception handling statements for JavaScript.

```
function SetFullName(FirstName,LastName)
{
        if (FirstName != null || LastName != null)
        {
                FullName=FirstName + " " + LastName
                return FullName
        }
        else
        {
                throw "InvalidName"
        }
}
try
{
// statements to try
 var MyName=SetFullName(First, Last) // function could throw exception
}
catch (e)
{
 MyName="unknown"
 logMyErrors(e) // pass exception object to error handler
}
```

For detailed information about exception handling, refer to

http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide

See Statements – Exception Handling Statements section.

Chapter 4

Where can I use JavaScript in Service Manager?

JavaScript can be used in several places throughout Service Manager. These areas focus on providing maximum flexibility for users to tailor their environment through the use of JavaScript in Service Manager.

Note: HP advises that you store JavaScript in the ScriptLibrary and refer to it from other places.

You can use JavaScript in these places in Service Manager.

- Cascade updates
- Display Screens
- Display Options
- Format Control
- Interoperability (the **ioaction** table)
- Links
- Processes (Document Engine)
- ScriptLibrary
- Scripts
- Triggers
- Wizards

What is the ScriptLibrary?

The ScriptLibrary is a file within Service Manager used to create, edit, compile, test, and store scripts and functions that can be called and executed from anywhere within Service Manager. The ScriptLibrary contains JavaScripts created by users. It is a central place for you to create, edit, compile, test, and store scripts and functions.

Some of these stored JavaScript programs can be used to interface with 3rd party Web Services providers, such as real-time currency conversions. All of these JavaScripts are identified as SOAP packages within the ScriptLibrary. Integrations are typically performed through JavaScript stored in the Script Library as well.

Note: Do not use Adminstrative Mode when entering the ScriptLibrary, since functions such as compile and execute are not available in that mode.

You can access any script or function in the ScriptLibrary from any of the locations in HP Service Manager that support JavaScript. To call one of these functions, use one of the following statements:

```
system.library.<scriptname>.<functionname>( parameters )
```

or

```
library.<scriptname>.<functionname>( parameters )
```

ScriptLibrary editing environment

This is a list of guidelines for the size of scripts in the ScriptLibrary.

- The maximum size of a script allowed in the ScriptLibrary is 256K.
- The size value is controlled by the **recordsizelimit** parameter in the sm.ini file. This is general record size in database. The defaulting is 256K, but changing this value will affect all records in the database.
- It is a best practice to allow the **recordsizelimit** to default to 256K since SOAP scripts generated by HP's WSDL2JS utility can exceed 64K. Setting **recordsizelimit** to a smaller value can result in failure when using the WSDL2JS utility with complex files.

The color codes that the script editor uses are:

- Red: Keywords
- Green: Objects, methods, and properties as well as function names
- Teal: Comments
- Magenta: Curly braces, operators, punctuation characters
- Black: Default color

Calling a function in the ScriptLibrary

ScriptLibrary records contain functions that can be called from anywhere you can call JavaScript. Using this technique allows for more effective management of user JavaScript functions. Functions can use parameters as part of the calling structure, and they can also return values.

First, you must establish a record in the ScriptLibrary that contains the function to be called. Within the record you must know the name of the script as well as create a properly-constructed function.

Note: A record can contain more than one function.

This is an example of using more than one function.

```
/* Sample script system.library.functionTest */

function square(value)
{
```

```
        var ret_val = value * value;
        return ret_val;
}
function cube(value)
{
        var ret_val = value * value * value;
        return ret_val;
}
```

To call the function use the following syntax:

```
system.library.<script name>.<function name>(<Comma separated list of
parameters>)
```

The following is an example of calling the functions of system.library.functionTest:

```
var x = 3;
var y = system.library.functionTest.square(x);
print(x, "  squared =  ", y);

x = -2;
y = system.library.functionTest.square(x);
print(x, "  squared =  ", y);

x = 2;
y = system.library.functionTest.cube(x);
print(x, "  cubed   =  ", y);

/*
The function calls produce the following output:
 3  squared =  9
-2  squared =  4
 2  cubed   =  8
*/
```

When working with the ScriptLibrary, you can compile the JavaScript code to check for syntax errors. Executing the script will compile the code and then run it. Before executing the code, ensure that all inputs are available.

You can access any script or function in the ScriptLibrary from any tool within Service Manager that supports JavaScript, by using the following syntax:

```
system.library.<scriptname>.<functionname>( parameters )
```

Within a ScriptLibrary record, you can call any function that is defined previously within the same record by simply typing:

```
<functionname>( parameters )
```

To call JavaScript from any RAD process panel, you can use the jscall RAD function. The syntax for this function is:

```
jscall( "script.function" , arg1 , arg2, ... , argN )
```

The following statements illustrate a JavaScript call to the system library and a call to the script library via jscall(), respectively:

```
system.library.tzfunctions.getTZforOperator("falcon");
jscall("tzfunctions.getTZforOperator", "falcon");
```

Example: Function to control close button label in Change Management

Change Management uses the function `getCloseLabel()` to control what label to display on the Close button. The labels are Close Task, Close Change, or Close Phase. Included in this function is code to process a special case, Unplanned Changes, in the out-of-box system. This code can serve as an example for Administrators who need to customize the Close button label for other special cases.

The function is `getCloseLabel(filename, currentPhase, file)` in the ScriptLibrary file changeManagement. The details are as follows:

- function `getCloseLabel(filename, currentPhase, file)`
- filename: name of the table we are currently on, cm3r for Change record, cm3t for Task
- currentPhase: current phase of the change or task
- file: the file record

This function is currently called from display screen `cm.view.display`
Example:

```
$L.close.label=jscall("changeManagement.getCloseLabel", filename
($L.filed), current.phase in $L.file, $L.filed)
```

The log4js script

The log4js is a logger type script similar to log4j. It provides a lightweight but useful script to log events in a Service Manager script that you want to debug . It provides a means to trace information in the Message view while controlling the level of detail being logged. It allows developers to turn in-line debugging on and off.

The log4js script allows developers to set the following logging levels.

log4js log level	Description
OFF	Nothing is logged.
ALL	Everything is logged.
DEBUG	Debug information is logged.
INFO	Information messages are logged (default).
WARN	Warning messages are logged.

log4js log level	Description
ERROR	Error messages are logged.
FATAL	Fatal error messages are logged.

API Details

The log4js should be initialized before it is used.

Constructor syntax:

```
var samplelog = new lib.log4js($loglevel, $prefix);
```

You can also use the following method to create an instance of the logger:

```
var myLogger=new lib log4js.Log(lib.log4js.Log.INFO);
```

Sample code snippets

```
myLogger.info("an info");
myLogger.warn("a warning");
myLogger.error("an error");
```

You can change the log level or turn off logging with:

```
myLogger.setLevel(lib.log4js.Log.WARN);
myLogger.setLevel(lib.log4js.Log.OFF);
```

You can also change the log level or turn off logging with this format:

```
myLogger.setLevel("WARN");
myLogger.setLevel("OFF");
```

The log4js script supports appending a prefix to each message. For example:

```
var myLogger=new lib.log4js.Log("INFO","my prefix");
```

Type of Logger:

Logger	Description
alertLogger	Output information to Service Manager Message view or the server log file (default: <SM server installation directory>\logs\sm.log), depending on whether a background or foreground process calls the log4j functions. background: server log foreground: Message view

API methods:

The following table lists the API methods for log4js.

Method	Description
$log.debug	Write a debug message.
$log.info	Write an information message.
$log.warn	Write a warning message.
$log.error	Write an error message.
$log.fatal	Write a fatal message.
$log.setLevel	Set a log level.
$log.getLevel	Retrieve the log level.
$log.setPrefix	Set the prefix information.
$log.getPrefix	Get the prefix information.
$log.getLogger	Get logger information.
$log.setLogger	Set the logger information.

Script name: log4js

Package name: null

Performance impact

The log4js uses the standard Service Manager print method to write the Service Manager server log file or the Messages view, which is useful for debugging your JavaScript, but keep in mind that when logging is turned on there is a performance impact so log4js should be used carefully.

Examples of calling JavaScript from different Service Manager tools

The following examples offer suggestions about using JavaScript in some of the tailoring tools that support this functionality. Additional JavaScript examples can be found in section "JavaScript and Service Manager reference" on page 203.

Format Control

The following example can be used to verify whether or not the operator who was just entered in the assignee.name field in the incident ticket is currently logged in. Otherwise, validation will fail.

1. In the Calculations section of Format Control, in the initial column enter a condition of true. Then add the following in the calculation column:

```
$assignee.old = nullsub(assignee.name in $file, "no assignee")
```

2. Click the JavaScript tab, enter true for Add and Update, and enter the following statements:

```
var usersXML = system.users;
var usersString = usersXML.toXMLString();
var file = system.vars.$file;
var newAssignee = file.assignee_name;
system.vars.$IsUser=usersString.indexOf(newAssignee);
```

3. In the Validation section enter the following, in the Add and Update column:

```
assignee.name in $file~=$assignee.old;
```

4. And the following expression in the Validation column:

```
$IsUser > 0
```

5. Enter the following in the Validation message field:

   The new assignee is not logged in at this time. Please try another assignee.

This example uses the standard indexOf method for the String Object, and the toXMLString() function, to search for the user who was entered as the new assignee in the XML object containing all logged-in users.

Any RAD variable can be set and used by preceding it with system.vars. This lets variable values be exchanged between JavaScript and RAD expressions.

Links

This example shows how to query a record from within Service Manager, and how to update fields in that record. Additionally, it shows how to transfer an array from Service Manager to JavaScript. An array from Service Manager cannot be directly copied into a JavaScript array, because the syntax of Service Manager {"IM1001", "IM1002"} does not work in JavaScript. Properties such as the length property assume that the value is a function if it is enclosed in curly braces {}. To successfully convert a Service Manager array into a JavaScript array, you can either use the SCFile.toArray method, or create a "while" loop or a "for" loop to move through the elements and do an element-by-element transfer, as shown above.

Note: In JavaScript, it is very important that you handle null values. Functions such as length may not work on null values.

In this example, the JavaScript code:

- Queries the configuration item that was previously selected from a fill.
- Increases a counter in that configuration item called num.issues.on.CI (this new field has to be added in the device file).
- Adds the new ticket number to an array of issues in the device file called array.incidents.
- Updates the device record.

- If the configuration item does not exist, it issues an error message.
- If the item is currently down, it sends an alert to the person who is opening the ticket.

```
var CI_Record=new SCFile("device"); /* opens the device file */
var CIName=system.vars.$File.logical_name; /* passes in the logical
name from $File */
/* Selects the configuration item to update */
function getConfigurationItem( CIName )
{
var findCI_RC = CI_Record.doSelect( "logical.name=\""+ CIName + "\""
);
if (findCI_RC==RC_SUCCESS)
{
return CI_Record;
}
else
{
print( "Could not find configuration Item. " +
RCtoString( findCI_RC ) );
return null;
}
}
/* Increases the use counter that counts how often this item was
affected by an incident ticket and enters the incident ticket number
into the array of incidents */
function increaseUseCounter(CI)
{
if (CI.is_down == true)
{
print("The configuration item is down at the moment. Please check
existing tickets if the cause for this issue is related to the down
system.");
}
CI.num_issues_on_CI++; /* increase counter */
var incNumber = system.vars.$File.number; /* get the incident ticket
number */
var array_incidents = new Array();
var ind = 0;
while(CI.array_incidents[ind] != null) /* convert the SC array into a
JS array (see paragraph below) */
{
array_incidents[ind]=CI.array_incidents[ind];
ind++;
}
var j=0;
var temp_string = array_incidents.toString();
var i = temp_string.indexOf(incNumber);
if (i < 0) /* Check if incident is already in the list */
{ /* If not, enter the incident number to the list */
if (array_incidents == null)
```

```
{
j=0;
}
else
{
j=array_incidents.length;
}
array_incidents[j] = incNumber;
CI.array_incidents = array_incidents;
}
CI.doUpdate();
}
/* Call previously defined functions to execute code */
CI=getConfigurationItem(CIName);
increaseUseCounter(CI);
```

Trigger scripts

The triggers.g form contains a field for entering JavaScript (Script). HP Service Manager does not allow you to compile JavaScript on this form, thus syntax errors are not caught until execution. HP recommends that the **Script** field contain single statements that invoke functions in the ScriptLibrary. For example, `return system.library.myTriggers.triggerXYZ( );`.

Use the Compile icon in the ScriptLibrary to verify the syntax of your trigger script function. A trigger script only needs to return a value if the trigger fails. The user can indicate the failure of a trigger by returning the value of –1. Trigger scripts can access the value of the current record in memory using the global object called system.record, and they can access the value of the same record on disk using the global object called system.oldrecord. For example, `return system.library.myTriggers.triggerXYZ( system.record, system.oldrecord );`.

Triggers example

In this example, a trigger sends a message to the assignment group identified in an incident record that the record has been updated and to whom it is assigned. It then adds an activity note that the message has been sent.

```
var NewRecord=system.vars.$L_new; /* This is the $L.new record passed
in from the trigger */
var incNumber=NewRecord.number;
var incAssignee=NewRecord.assignee_name;
if (incAssignee==null)
{
       incAssignee="no one at the moment";
}
/* Insert a message schedule record into the schedule file */
function insertSchedule( incNumber, incAssignee, Adressees)
```

```
{
        var newSchedule = new SCFile( "schedule" );
        var today=new Date();
        newSchedule.application= "message.bg";
        newSchedule._class = "problem";
        newSchedule.name= "message processor record";
        newSchedule.expiration= today;
        newSchedule.strings[1]="1";
        newSchedule.strings[2]="problem update";
        newSchedule.strings[3]="Problem " + incNumber + " has been
        updated. It is assigned to " + incAssignee + ".";
        newSchedule.strings[5]="pm.main";
        newSchedule.strings[6]=" Soap-Windows XP";
        newSchedule.strings1=Adressees;
        var rc = newSchedule.doInsert();
}
/* Selects the operators that are members of the assignment group to
return to the calling function */
function selectAddressees(assignmentGroup)
{
        var assignment = new SCFile( "assignment" );
        var findAssignee = assignment.doSelect( " name=\""+ assignmentGroup +
        "\"" );
        if ( findAssignee == RC_SUCCESS )
        {
                addressList=assignment.operators;
                return addressList;
        }
        else
        {
                print( "Could not find assignment group. " +
                RCtoString( findAssignment ) );
                return null;
        }
}
/* Inserts an activity with the information that the email
notification
 to the assignment group members has been sent */
function addActivity(incNumber)
{
        var newActivity = new SCFile( "activity" );
        var today=new Date();
        newActivity.number= incNumber;
        newActivity.datestamp= today;
        newActivity.type= "Update";
        newActivity.description[0]= "Notification email on the update has
been
        sent to all members of the assignment group.";
        var result = newActivity.doAction("add");
        if (result == RC_CANT_HAVE || result == 51 )
```

```
        {
                system.library.activityUpdates.scheduleActivityUpdate
                ( newActivity, newActivity.type, newActivity.description );
        }
}
/* Selects the operators to send the message to */
Addressees = selectAddressees( NewRecord.assignment );
if (Addressees != null)
{
/* adds the message to the schedule file for sending */
insertSchedule( incNumber, incAssignee, Addressees);
/* adds an activity to inform the user of the email being sent*/
addActivity(incNumber);
}
```

The example also shows (among other things) how to call a Document Engine Action, how to call a function from the Script Library, and how to insert a new record. In addition, it shows how to use several functions within a single JavaScript and how to call them.

Note: Because the class field name in the schedule file is in conflict with the class keyword used in JavaScript, it must be preceded by an underscore (_).

Using RAD functions in JavaScript

You can use most RAD functions in JavaScript. To use a RAD function in a JavaScript, simply prefix the function with "system.functions" and replace any periods (.) in the function name with an underscore (_).

For an example of how to do this, compare the following functions:

- "RAD function: parse" on page 116
- "RAD function: evaluate" on page 93
- "JavaScript function: parse_evaluate()" below

Refer to "List: RAD functions" on page 79 for a list of other RAD functions.

JavaScript function: parse_evaluate()

Returns the value of a given expression.

Syntax

```
system.functions.parse_evaluate( exp , 10 );
```

Note: The data is the character string to parse and the parse type is the numeric value normally passed into the parse function.

Example

If in the useroptions table there is an expression $lo.username="Sys.Admin," this example returns the value of that expression.

```
if (system.functions.parse_evaluate(expression in $l.file, 10 ))
   userIsSysAdmin = true;
```

Note: The expression works when the statement being evaluated and parsed contains thread or global variables. It does not work for local variables. However, you can transfer local variables to thread variables before calling the parse_evaluate expressions.

RAD functions: parse() and evaluate()

Returns the value of a given expression.

Syntax

```
evaluate(parse(exp, 10 )
```

Note: The data is the character string to parse and the parse type is the numeric value normally passed into the parse function.

Example

If in the useroptions table there is an expression $lo.username="Sys.Admin," this example returns the value of that expression.

```
if (evaluate(parse(expression in $l.file, 10)))
   $userIs.Sys.Admin = true
```

Note: The expression works when the statement being evaluated and parsed contains thread or global variables. It does not work for local variables. However, you can transfer local variables to thread variables before calling the parse and evaluate expressions.

JavaScript global properties

Global properties are predefined properties that are not part of any object. They are names representing values that are globally defined. The core JavaScript language has several global properties, such as NaN (Not a Number), and Service Manager has a number of additional global properties.

There are two categories of Service Manager-defined global properties:

Query operators

JavaScript return code properties

Query operators

These operators are used solely for constructing query expressions using the Query object.

Note: The Query object has been deprecated, but older JavaScript code within the ScriptLibrary still uses the object.

Query operator	Description
EQ	Equals to
LIKE	Starts with
NEQ	Not equal to
GT	Greater than
GE	Greater or equal to
LT	Lower than
LE	Lower or equal to
ISIN	Is an element in the array

Example:

```
var q=new SCFile("probsummary")
q.doSelect(new QueryCond("number", EQ, "IM10002"));
```

JavaScript return code properties

Use these properties (which are represented internally as integer values) to test the values returned by object methods such as SCFile. To print the error message rather than the return code, use the RCtoString function.

Return code	Description
RC_ERROR	Some other error occurred. Examine the contents of the Messages view or the sm.log file for more information.
RC_SUCCESS	The operation succeeded.
RC_CANT_HAVE	The operation failed because the resource is unavailable because some other user or process has the resource locked.
RC_NO_MORE	No more records are available in the result set.
RC_DUPLICATE_KEY	The insert operation failed because the file already contains a record with this unique key value.
RC_MODIFIED	The update operation failed because the record was modified by another user or process since you read it.
RC_DELETED	The operation failed because the record was deleted by another user or process.
RC_BAD_QUERY	The query failed due to incorrect query syntax.

Return code	Description
RC_NOT_ AUTHORIZED	The request operation was not performed due to an authorization failure. Check the permissions associated with the user who submitted the request.
RC_VALIDATION_ FAILED	The operation failed because the data supplied in a field or the record did not pass validity checks performed by the application.
RC_UNABLE_TO_ WRITE_TO_FILE	The operation failed. Check to make sure the file name is correct, that the file exists, and that it is not read-only.
RC_UNABLE_TO_ CLOSE_FILE	The operation failed. Check to make sure the file name is correct, that the file exists, and that it is not read-only.
RC_UNABLE_TO_ DELETE_FILE	The operation failed. Check to make sure the file name is correct, that the file exists, and that it is not read-only.
RC_INVALID_ FILENAME	The operation failed because the file name is not valid.

Example:

```
var rc = mySCFile.doInsert();
if ( rc == RC_SUCCESS )
{
       print( "Insert succeeded" );
}
else
       throw( "Error " + RCtoString(rc) + " occurred trying to insert a
       record" );
```

Order of execution of JavaScript versus RAD language

This section lists all tailoring tools in which JavaScript can be directly entered and describes at what point in the process the JavaScript code will be executed.

- Format Control: The JavaScript is executed after calculations and before validations. JavaScript code in Format Control is executed via the format.cjavascript RAD.
- Links: JavaScript in links is executed after the (initial) expressions, but before the fill or find is executed. This is done in us.link on evaluate.expressions.
- The post JavaScript is executed after the fill or find was performed, immediately after the post expressions. This happens in us.fill or us.find, respectively, on the eval.post.expressions panel.
- Triggers: The JavaScript in the triggers file is executed after the RAD application. This is controlled by the server binaries and not by RAD code.

- Cascade updates: The JavaScript in cascade updates is executed from the RAD application process.update.config.record. The evaluate expressions panel is executed before the evaluate.javascript panel, so that expressions take precedence over JavaScript.
- Scripts: In script.execute on panel exec.statements, RAD expressions are processed before JavaScript expressions.
- Wizards: Wizards use JavaScript in 3 different areas: During File Selection, Actions, and Cancel Expressions.
  - File Selection: The Select $L.file by information is processed first, followed by the Initial Expressions and then the JavaScript.
  - Actions: In the Actions tab, the Expressions are run first, then the JavaScript, followed by the information on the Format Control / Process Name tab.
  - Cancel Expressions: First the Expressions Executed on Cancel are processed, followed by Javascript Executed on Cancel.
- Interoperability (ioaction file) : The ioaction file executes only JavaScript and not RAD expressions. The JavaScript code is executed in the ioevents.process.action application on the panel process.action.
- Display screens: In the display application on the panel prep.screen, the JavaScript expressions are executed after the RAD expressions.
- Display options: Display options have pre- and post-JavaScript and RAD expressions that are executed in the following order:

1. Pre-RAD expression
2. Pre-JavaScript
3. RAD application
4. Post-RAD expression
5. Post-JavaScript
6. Display Action

- Processes: Process records have initial and final JavaScript and RAD expressions executed in the following order:

1. Initial RAD expression
2. Initial JavaScript
3. Pre-RAD expressions
4. RAD application(s)
5. Post-RAD expressions
6. Final RAD expression
7. Final JavaScript
8. Next Process (if applicable)

- Schedule: JavaScript in the schedule record is executed after the RAD application is called and before the check for deleting or rescheduling the schedule record.

Corresponding JavaScript syntax for RAD language tasks

Most Service Manager tasks can be translated from RAD language to JavaScript by using the JavaScript expressions identified in the following table. In the JavaScript expression, the ObjectName is the name of a Service Manager object such as a table.

Service Manager task	RAD panel / expression	JavaScript expression
Create a file variable	rinit	new SCFile(“filename”)
Select a record	select	RC=ObjectName.doSelect (“Query”)
Update a record	rupdate	RC=ObjectName.doUpdate()
Add a record	radd	RC=ObjectName.doInsert()
Delete a record	rdelete	RC=ObjectName.doRemove()
Refer to a field in a file	result=fieldname in filevariable	result=ObjectName.fieldname
Access an element of an array	result=rownumer in arrayname in filevariable	result=ObjectName.arrayname [rownumber]
Send a message to screen	call message application	print(“message”)
Convert a field / Object to string	str()	RC=ObjectName.getText()

Chapter 5

System Language reference

This is the reference section for the system language. In this section you will find a list of the RAD functions, a list of the rtecalls, and a list of RAD routines.

Deleted RAD applications

The following RAD applications have been deleted from Service Manager 9.31 because the functionality they provided is no longer supported by Service Manager, or it has been replaced with updated capabilities.

add.bigroup	apm.get.incident.query	cm.open.scm7
adl.initial.setup	apm.get.initial.query	cm3r.group.assign.totals
ag.editor.backup	apm.get.parts	cm3r.group.request.totals
agmap.flow	apm.get.probassign.query	cm3t.group.assign.totals
agmap.flow.panels	apm.get.problem.query	cm3t.group.request.totals
agmap.panel.xref	apm.get.values	convert.db2mvs
agmap.print.arrays	apm.inbox	convert.oracle
agmap.print.condense	apm.initial	convert.sqlserver
agmap.print.condense.only	apm.launch.update	convert.sybase
agmap.print.fields	apm.monthly.downtime.job	create category
agmap.process.reports	apm.open.inbox	db2mvs.t-emplate.to.sqlbasename
am.cigroup.bulkupdate2	apm.post.cost.save	display.backup
am.search.related.SAV	apm.remove.membership	display.pass.file.position
apm.add.assign.group	apm.remove.record	encl.appl.fla
apm.add.group	apm.rename.group	find.alias.fields
apm.add.parts	apm.repair.problem	find.illegal.fc
apm.add.profile	apm.set.assignment	fix.form.graph

apm.add.user	apm.tapi.header	get.serno
apm.admin	apm.thread.stub	isg.check.pm.contract
apm.agi.main	apm.user.to.group	isg.decide.contract
apm.assign.clean	append.to.related	isg.delete.sm.warnings
apm.build.child.list	appl.copy.rename	isg.set.warnings
apm.build.screlate	big.green.arrow	label.display
apm.build.screlate.sub	category.delete	login.saf.msgs
apm.ca-scade.problem.close	cau.add.profile	pm.close.pending
apm.cp.send	cau.add.profile2	pm.log
apm.create.group	cau.add.user	pmc.sql.repair
apm.create.record	cau.add.user.backup	rca.add.group
apm.delete.group	cau.check.group	rca.add.profile
apm.delete.membership	cau.edit.profile	rca.add.user
apm.delete.profile	cau.get.rm.profiles	rca.admin
apm.delete.record	cau.main.2	rca.create.group
apm.delete.user	cau.remove.from.groups	rca.delete.group
apm.done.link	cau.r-emove.from.msg.groups	rca.delete.profile
apm.dt.process.demon	cau.remove.profile	rca.delete.user
apm.edit.assign.group	cc.add.group	rca.edit.group
apm.edit.group	cc.add.profile	rca.edit.profile
apm.edit.operator	cc.add.user	rca.edit.user
apm.edit.profile	cc.admin	rca.fill.profile
apm.edit.record	cc.create.group	system.nls
apm.edit.user	cc.delete.group	trigger.counter
apm.email.problem	cc.delete.profile	trigger.problem.homesite
apm.end.user	cc.delete.user	trigger.problem.shadowsite

apm.expand.downtime	cc.edit.group	unconvert.qbe.lists
apm.fill.profile	cc.edit.profile	us.calc.delta
apm.find.next.event	cc.edit.user	us.gen.message
apm.fix.call	cc.end.user	us.load.errmsg
apm.fix.shell	cc.remove.membership	us.make.text
apm.fix.stored.queries	cc.required.text	xanadu.timezone.set
apm.get.assignments	cc.user.to.group	

List: RAD functions

A RAD function is an operand, which means that it can be used in any expression on the right hand side of an equals sign. Functions provide a method of performing certain commands that will return a value when executed. For example:

- operator()--returns the logged-on operator ID.
- tod()--returns the current date and time.
- option()--returns GUI option number of the last displayoption selected.

RAD functions can be used in statements to further define selection criteria and/or execute commands, initialize values, and perform calculations. Service Manager syntax rules must be followed. For example, the result of a RAD function can be assigned to a variable using the assign (=) assignment statement: `$string.length=lng($string)`

In most cases, there is a JavaScript function that can do the same operation as a RAD function. If you prefer to work with JavaScript rather than RAD, see "Using RAD functions in JavaScript" on page 71 for more information.

Function	Description
add.graphnodes(xml)	Returns XML to the Run-Time Environment (RTE) for the expand action on a node in the graph diagram.
cache.flush	Removes the specified item from the memory cache.
cleanup	Frees the storage associated with a variable.
contents	Returns a structure containing the current record in a file variable.
copyright	Returns a string with the HP copyright notice.
currec	Represents the current file handle in queries.
current.device	Returns a string with the type of system the user is using to log in to

	Service Manager, such as "SOAP-Windows Vista."
current.format	Returns a string containing the name of the current format.
cursor.field.contents	Returns a string containing the contents of the input field in which the cursor was positioned when the last interrupt key was pressed.
cursor.field.name	Returns a string containing the name of the field in which the cursor was positioned when the last interrupt key was pressed.
cursor.field.name.set	Moves the cursor to a field.
cursor.field.readonly	Returns true if the selected field is read-only. Returns false if the selected field is not read only.
cursor.line	Returns the number of the line, relative to the screen, in which the cursor was positioned when the last interrupt key was pressed.
date	Returns the date portion of a date/time variable and defaults time portion to '00:00:00'.
day	Returns the day of the month for a date regardless of the date format
dayofweek	Returns a number from 1 to 7 representing the day of the week for a specific date.
dayofyear	Returns the day of the year for a date regardless of the date format.
delete	Returns an array with specific elements deleted.
denull	Returns an array with all trailing NULL entries deleted.
descriptor	Returns the database dictionary descriptor record for a file variable.
evaluate	Executes a specific string as though it were a processing statement.
evaluate.query	Evaluates a query against a record.
exists	Checks for the existence of a field in a file.
fduplicate	Copies an entire file variable from one record to another.
filename	Returns a string containing the name of the file for a specified file variable.
frestore	Restores all fields in a file variable to their original database values.
genout	Generates a formatted string based on a record and format name passed as parameters.
get.base.form.name	Returns the base name of a form, stripping off the form extension for GUI or Web forms.
get.dateformat	Returns the date format of the current operator ID.
get.graph.action	Returns the activated action on a node in the graph diagram.

get.graph.id	Returns the value of the graph ID for the expand action on a node in the graph diagram.
get.graphnode.id	Returns the value of the node ID for the expand action on a node in the graph diagram.
get.graph.target	Returns the target for an activated action on a node in the graph diagram.
get.lock.owner	Returns the name of the owner of a resource lock.
get.timezoneoffset	Returns the absolute time difference between GMT and the time zone of the operator.
get.uid	Returns a 24-character identifier that is unique to an installation of Service Manager
gui	Determines whether or not Service Manager is running in GUI mode.
index	Returns the element number of an array or the position in a string that matches a specified string.
insert	Returns a modified array with a specified number of new elements inserted at a specified location.
iscurrent	Determines if the record with which you are working is identical to the version stored in the database.
jscall	Enables you to invoke JavaScript from within RAD.
lioption	Indicates whether a given input value is a licensed option on the current system.
lng	Returns the number of elements in an array or characters in a string.
locks	Returns array of current outstanding locks.
logoff	Logs a user off.
logon	Logs a user on.
mandant	Allows an application to create a subset of the current Mandanten values.
max	Returns the largest value in a list of values or in an array.
messages	Provides logging functions for use in the memory message log. The log is implemented as a wrap-around cache of the x most recent messages (error, informational, and action).
min	Returns the smallest value in a list of values or in an array.
modtime	Returns the time a record was last modified.
month	Returns the month of year for a date regardless of the date format.

multiselect.selection ("fieldcontents")	Returns the contents of the fields associated with the multiselection.
multiselect.selection ("fieldname")	Returns the field name of the column associated with the multiselection. The associated column name is set using the Forms Designer property "Selection Field."
multiselect.selection ("rows")	Returns an array of row numbers that were selected.
multiselect.selection ("selected")	Returns true or false if the provided file variable has multiple records selected.
multiselect.selection ("selections")	Returns the number of rows selected.
multiselect.selection ("tablename")	Returns the Table ID associated with the multiselection.
null	Returns true if the value of a variable is NULL. Returns false if the value of a variable or field is not NULL.
nullsub	Substitutes a null field with the value given.
operator	Returns the string of the name of the currently logged on operator.
option	Returns the number of the last option key pressed.
parse	Returns an evaluative expression from a given string value.
perf	Evaluates system performance and writes the information to disk.
policyread	Reads the data policy from a datadict table.
processes	Returns array of current logged on processes.
prof	Returns the value requested on various system performance profiles.
recordcopy	Copies a set of fields from one record to another record.
recordtostring	Takes the value of a field from an array and appends the value to a string.
round	Returns a number rounded to a specified number of digits.
same	Compares two compound or null values. Returns true if two values are identical, otherwise returns false.
scmsg	Returns a text string matching the format of the message ID number and message class specified. The function replaces any variables in the message with the text specified in an array of values.
set.timezone	Sets time zone and other system parameters.
setsort	Sorts the fields of an array in a file.

shutdown	Shuts down Service Manager from inside the system. This function does not return any values.
simple file load	loads an unload file containing only data records of one file into a target file.
str	Returns a string of a string or non-string data variable.
stradj	Makes a string a specific length by either clipping or adding trailing blanks.
strchrcp	Replaces part of a string with copies of another string.
strchrin	Inserts copies of a string into another string.
strclpl	Clips a number of characters from the beginning of a string.
strclpr	Clips a number of characters from the ending of a string.
strcpy	Replaces part of a string with a substring.
strdel	Deletes a number of characters from a specific spot in a string.
strins	Inserts a substring into another string.
strpadl	Pads a string with leading blanks until the string is a certain size.
strpadr	Pads a string with trailing blanks until the string is a certain size.
strrep	Returns a string, replacing a specified character sequence with another string.
strtrml	Removes all leading blanks from a string.
strtrmr	Removes all trailing blanks from a string.
substr	Returns a string from a portion of a string.
substrb	Extracts a substring from a multibyte character or binary string.
sysinfo.get (“ActiveFloatUsers”)	Returns the total number of currently active floating users.
sysinfo.get (“ActiveLicenses”)	Returns the total number of currently active licenses.
sysinfo.get (“ActiveNamedUsers”)	Returns the total number of currently active named users.
sysinfo.get ("AuthMode")	Returns the authentication mode of the current user.
sysinfo.get (“ClientNetAddress”)	Returns the Service Manager client network IP address as a character string.
sysinfo.get (“ClientOSName”)	Returns the name of the operating system that Service Manager is running on.

sysinfo.get (“ClientPID”)	Returns the PID for RAD.
sysinfo.get (“ClientVersion”)	Returns the version number of the client software as a string.
sysinfo.get(“Display”)	Returns “GUI” or “text”, depending on the type of client that is running.
sysinfo.get (“Environment”)	Returns the name of the environment in which the client is operating.
sysinfo.get (“languagecode”)	Returns the server session language code value that is used for localized forms and messages.
sysinfo.get (“MaxFloatUsers”)	Returns the total number of floating users logged on since the server started.
sysinfo.get (“MaxLicenses”)	Returns the maximum number of licenses used since the server started.
sysinfo.get(“Mode”)	Returns a value that states the type of client being used.
sysinfo.get (“PrevLabel”)	Returns the label name of the last RAD panel that was executed.
sysinfo.get (“PrintOption”)	Returns printing option information, “server” or “client” depending on how the system is set.
sysinfo.get(“Quiesce”)	Returns “true” if system is quiesced and “false” if system is not quiesced.
sysinfo.get(“RecList”)	Returns “true” if record list is enabled and “false” if record list is disabled.
sysinfo.get (“ServerNetAddress”)	Returns the Service Manager server network IP address as a character string.
sysinfo.get (“ServerNetPort”)	Returns the Service Manager server port used by the connected client or the system name when called by a background process.
sysinfo.get (“ServerPID”)	Returns the PID of the server as a number.
sysinfo.get (“Telephony”)	Returns “true” if telephony is enabled and “false” if telephony is disabled.
sysinfo.get (“ThreadID”)	Returns the number of the current RAD thread.
sysinfo.get (“TotalFloatUsers”)	Returns the total number of floating users allowed.
sysinfo.get (“TotalLicenses”)	Returns the total number licenses allowed.

sysinfo.get (“TotalNamedUsers”)	Returns the total number of named users allowed.
sysinfo.get (“TotalProcs”)	Returns the total number of executing system and user tasks.
sysinfo.get (“TotalSystemProcs”)	Returns the total number of system tasks.
sysinfo.get (“TotalUserProcs”)	Returns the total user tasks.
time	Returns the time portion of a date/time variable.
tod	Returns the current date and time.
tolower	Returns a string, replacing upper-case letters with lower-case letters.
toupper	Returns a string, replacing lower-case letters with upper-case letters.
translate	Returns a string translated from one character set to another.
trunc	Truncates the decimal digit number to a specified number of digits.
type	Returns the numeric type of its argument.
val	Returns a non-string data type converted from a string.
version	Returns a list of version information.
year	Returns the full year for a date regardless of the date format.

RAD function: add.graphnodes(xml)

Description

A RAD function that returns XML to the Run-Time Environment (RTE) for the expand action on a node in the graph diagram.

Function

add.graphnodes()

Example

```
add.graphnodes(xml)
```

RAD function: cache.flush

A RAD function that removes the specified item from the memory cache.

Function

cache.flush

Format

cache.flush(*nType*, *pItem*)

Parameters

nType is a number from 1 to 4 representing the type of flush operation. Flush types 1, 2, and 4 remove the item from the global shared memory cache. Flush type 3 removes the item from the local format cache.

- 1 flushes application/code records
- 2 flushes dbdicts
- 3 flushes formats
- 4 flushes SQL maps

pItem is the name of the item to flush enclosed in quotation marks. Note that if flush type 3 does not use the *pItem* parameter.

Factors

This function returns no value.

Example

To flush the dbdict record for the probsummary table:

```
cache.flush( 2, "probsummary" )
```

To flush the local format cache:

```
$L.type = 3
cache.flush( $L.type )
```

RAD function: cleanup

A RAD function that frees the storage associated with a variable.

Function

cleanup

Format

cleanup(file variable), where file variable is the variable you want to cleanup.

Factors

This function does not return a value. It cleans up (empties) the variable or field specified by setting the value to NULL.

You can use this function on any variable type (local, thread, or global), but it is generally used on local and thread variables to clean up values before a user logs out.

This is the preferred way to clean up both variables and fields in records as opposed to $x=NULL or field in $file=NULL.

Unless you run this function, variables remain in memory until their defining environment ends. For local variables, this is the parent RAD program. For thread variables, this is the parent RAD thread. For global variables, this is the user session.

Note: This RAD function is not available as a JavaScript system.functions call.

Example

```
cleanup($system)
```

RAD function: contents

A RAD function that returns a structure containing the contents of the current record in a file variable.

Function
contents

Format
contents(file variable), where file variable is the contents you want to obtain.

Factors
If you use the contents() function to compare two records, always use the denull function on both records first to remove null values. If one record is displayed and the other is not, the arrays in one record may contain extra null values.

Using the denull function on an empty array of structures converts the array of structures to a simple array.

Note: Always test for null values first, as shown in the following example:

```
denull(contents($save.rec))=denull(contents($rec))
```

Example
`contents($file.variable)`

RAD function: copyright

A RAD function that returns the HP copyright notice in a string.

Function
copyright

Format
copyright()

Example
`copyright()` returns *Copyright© 1994-2011Hewlett-Packard Development Company, L.P.*

RAD function: currec

A RAD function that represents the current file handle on queries. When used as a select statement, it selects all records modified since the date you specify from the current file.

Function
currec

Format
currec()

Example
`modtime(currec())>'1/1/05'`

RAD function: current.device

A RAD function that returns a string containing the name of the device or Configuration item (CI) associated with the user that is currently logged on.

Function
current.device

Format
current.device()

Factors
The value of $lo.device is set to the value of current.device upon login.

Example
`current.device()` returns */dev/tty00* (on Unix)

RAD function: current.format

A RAD function that returns a string containing the name of the most recently displayed format.

Function
current.format

Format
current.format()

Example
`current.format( )` returns *problem.open*

RAD function: cursor.field.contents

A RAD function that returns a string containing the contents of the field where the cursor was last positioned when the user performed an action such as clicking Fill or Find, or double-clicking on a QBE list. If the action opens a new form, cursor.field.contents will return the focus on the pop up form.

Function
cursor.field.contents

Format
cursor.field.contents()

Factors
If the field is an array, the content of the current element is returned.

Example
`cursor.field.contents()` returns *a101a01*.

RAD function: cursor.field.name.set

A RAD function that sets the cursor to a specific field on a form.

Function
cursor.field.name.set

Format
cursor.field.name.set($field.name, $row.number)

Parameters
$field name is the field name
$row.number is for arrays (optional)

Factors

- This function returns no value. It is not necessary to use it in an assignment statement.
- If the field is an array, you can position the cursor on a specific row by specifying *$row.number.*

Example
`cursor.field.name.set("address", 2)`
After execution, the cursor appears in the second row of the *Address* input field on the next form displayed.

RAD function: cursor.field.name

A RAD function that returns a string containing the name of the input field where the cursor was located when the user pressed an interrupt key, such as the Enter or option keys. To position the cursor in a specific field, use the cursor.field.name.set function. The function returns the field name as it is defined in the current form.

Function
cursor.field.name

Format
cursor.field.name(n)

Factors
Returns the fully-qualified field name if the function is called with an argument.

Example
`cursor.field.name()` returns *number*.
`cursor.field.name(1)` returns *header,number*.

RAD function: cursor.field.readonly

A RAD function that returns true if the cursor is in a read-only field and false if the cursor is in a field that is not read-only.

Function
cursor.field.readonly

Format
cursor.field.readonly()

RAD function: cursor.line

A RAD function that returns the line number where the cursor was positioned when the user performed an action such as clicking Fill or Find, or double-clicking an item in a QBE list. If the action opens a new form, cursor.line returns the focus on the pop-up form.

Function
cursor.line

Format
cursor.line(n)

Factors

- $l=cursor.line() = the numeric value of the line where the cursor was positioned.
- $l=cursor.line(1) = the numeric value of the index of the array or structure where the cursor was last located, including consideration for vertical scrolling. If the cursor is in an array input field, *$l* contains the numeric value of the index of the array in which the cursor is positioned. If the cursor is not in an array input field, *$l* contains the numeric value, relative to the format in which the cursor is positioned, including accounting for vertical scrolling, which may be greater than the client's screen length.

RAD function: date

A RAD function that returns the date portion of a date and time variable.

Function
date

Format
date($date.time.variable)

Factors

- The date portion of a date and time value is midnight of that day, for example the time is *00:00:00*. The function uses the local time zone to determine the date.
- The argument can be any absolute date and time value, or an absolute time, including tod.

Example

```
$date.opened=date($date.time.variable)
```

Where the *$date.time.variable* contains the current date and time or another valid date and time value. For example, if $date.time.variable = *'08/01/08 10:10:10,'* then $date.opened = *'08/01/08 00:00:00'*
.

RAD function: day

A RAD function that returns the day of month for a date, regardless of the date format.

Function

day

Format

```
day=day($date)
```

Where the value of *$date* is a date and time value.

Factors

Although the set.timezone function affects the way a date is presented, the day function always extracts the proper day value.

Example

```
$mday=day('2/15/08')
```

After execution, the value of *$mday* is *15*.

RAD function: dayofweek

A RAD function that returns the day of the week for a date, regardless of the date format. The dayofweek function returns 1-7, where 1 is Monday, 2 is Tuesday, 3 is Wednesday, 4 is Thursday, 5 is Friday, 6 is Saturday, and 7 is Sunday.

Function

dayofweek

Format

```
day=dayofweek($date)
```

Where the value of *$date* is a date and time value.

Factors

The set.timezone function affects the way a date is presented, whereas the dayofweek function always extracts the proper day of week value.

Examples

From the RAD debugger type:

```
d dayofweek('11/06/09')
```

This expression displays the day of the week of a specific date. (The command 'd' stands for 'display.') In this example, the value of *dayofweek* is *5* or Friday.

From the RAD debugger type:

```
x $Ltoday=dayofweek('11/06/09')
```

This expression executes the function. (The command 'x' stands for 'execute.') In this example, you want to assign the value to a variable.

RAD function: dayofyear

A RAD function that returns the day of year for a date, regardless of the date format.

Function
dayofyear

Format

```
$yday=dayofyear($date)
```

Where: the *$date* variable is a date and time value.

Example

```
$yday=dayofyear('2/15/08')
```

After execution, the value of *$yday* is *46*

RAD function: delete

A RAD function that deletes one or more elements in an array and returns the updated array.

Function
delete

Format

```
delete(array, element number, [number of elements])
```

Where the *array* variable is the array, the *element number* is the index number of the first element to delete, and the *[number of elements]* is the number of elements to delete. The number of elements to delete is optional. The default value is one (*1*).

Example

```
delete({1,2,3},2)
```

Returns *{1,3}*

```
delete({1,2,3},2,2)
```

Returns *{1}*

RAD function: denull

A RAD function that compresses an array by removing all trailing NULL entries and returns the compressed array.

Function
denull

Format
denull(array)

Factors

- If the array contains a NULL entry in the middle of the array, that entry is *not* removed.

- Using denull on an empty array of structures converts the array of structures to a simple array.

  Note: Always test for NULL first.

- Displaying arrays extends the length of the array to accommodate window size.

  Tip: It is a good practice to denull arrays before records are added or updated. You can use `denull(contents())` to accomplish this, but you should first ensure that the contents do not include any empty array of structures or they will be converted to simple arrays.

Example

```
denull({1,2,3,,})
```

Returns *{1,2,3}*.

```
denull({1,,2,,})
```

Returns *{1,,2}*.

RAD function: descriptor

A RAD function that returns the database dictionary descriptor structure for the table specified.

Function
descriptor

Format
descriptor(*$filename*)

Parameters
The *$filename* variable is used for the table whose descriptor structure you are requesting.

RAD function: evaluate

Evaluates an operator and may return a value.

Function
evaluate

Format
$e = evaluate($x)

Factors

- An operator is created using the parse function.
- The evaluate function is also valuable when the statement to be evaluated is contained in a Service Manager field that is not qualified once the application is compiled.
- The parse flag is used to parse data in a form, which in turn converts the data to an operator (data type 10).
- Although an *lvalue* is always required, a *value* is not always assigned to the *lvalue*.

Example

`$x=parse("1+1",1)` returns two (*2*).
`$x=evaluate($x)`

RAD function: evaluate.query

A RAD function that evaluates a query against a record. Both are passed as function parameters.

Function
evaluate.query()

Format
$L.rc=evaluate.query(<query>, <file variable>)

Function returns
true - if the record matches the query
false - if it does not match the query
unkown - if a comparison in the query encountered a NULL value or a field was referenced in the query that does not exist in the record
null - if the function was called incorrectly

Parameters
The following parameters are valid for the evaluate.query function:

Parameter	Data type	Description
query"<query>"	operator or character	Can be a previously parsed query or a string containing a query
record"<file variable>"	record	Must be a valid file variable

Examples

```
$x = evaluate.query( "true", $L.file )
```

This example returns TRUE for any valid file variable.

```
$query = parse( "contact.name # \"S\" ", 4 )
$y = evaluate.query( $query, $L.filed )
```

This example returns TRUE only for contacts records that start with "S" and FALSE for every other contacts record.

RAD function: exists

A RAD function that checks for the existence of a field in a table.

Function
exists

Format
exists(<field name>, $file)

Factors

- Returns true or false.
- Replaces the `index(<field.name>, descriptor($file))>0` expression.
- The *$file* variable must be of data type **6**. Any other data type returns false.
- The first parameter can be either a character type variable or a quoted string.

Example

```
$L.return=exists("schedule.id", $L.schedule)
```

The value of *$L.return* is *true* if *$L.schedule* is a file variable containing a schedule record.

RAD function: fduplicate

A RAD function that copies an entire file variable from one record to another.

Function
fduplicate

Format
fduplicate($target, $source)

Factors

- The system does not recognize fduplicate alone.
- Returns a Boolean value: true if it is successful, and false if it fails.
- Used primarily in RAD process panels and in Format Control.

Example

```
$L.void=fduplicate($file0, $file)
```

Creates *$file0*, an independent copy of *$file*.

RAD function: filename

A RAD function that returns the name of the file for the specified file variable.

Function
filename

Format
filename(*$filename*)

Parameters
The *$filename* variable has been bound to a database file with the rinit command panel.

Factors
This function is useful when executing a common subroutine that has been passed a file variable. This function allows you to determine the file name because the local variable contains data from an unknown file.

Example
`filename($file)` returns the dbdict name.

RAD function: frestore

A RAD function that restores all the fields in a file to their original values from the database.

Function

frestore

Format

```
frestore($file)
```

Where:

The *$file* variable is the file for which you want to restore the fields to their original values.

Factors

- Use only with a Windows client.
- This function returns no value, it is not necessary to use it in an assignment statement.

Example

```
frestore($file)
```

If the *$file* variable is initialized and a record is selected, changes are made to the contents of the *$file*variable in memory, but the updates are not yet written to the database. After execution, the *$file* variable is restored to the original value, which is the state stored in the database.

RAD function: genout

A RAD function that generates a string containing the contents of a record using a specified form. The output produced is either fixed or variable length.

Function

genout

Format

genout($file.variable, $format.name)

Parameters

The *$file.variable* is a variable containing a record or records to process.

The *$format.name* variable is the name of format to be used.

Factors

- Spaces are substituted for NULL in fixed length output.
- All normal format processing occurs, including input and output routines. Data can therefore be automatically reformatted or manipulated as part of the function. For example, a date can be converted from *02/28/05 00:00:00*to *February 28, 2005.*

Examples

```
$output=genout($operator, "operator.view")

$output={"Name:joes Full Name:Joe Smith",

"Printer:sysprint Email:joes@hp.com"}
```

RAD function: get.base.form.name

A RAD function that returns the base name of a form, without the *.g* (GUI) or *.w* (Web) file extension.

Function
get.base.form.name

Format
$base.form.name=get.base.form.name($form.name)

Parameters
$form.name

Examples
In each of the following examples, the result is the same after execution: *$base.form.name* is *operator*.
Example 1:

```
$form.name="operator.g"

$base.form.name=get.base.form.name($form.name)
```

Example 2:

```
$form.name="operator.w"

$base.form.name=get.base.form.name($form.name)
```

Example 3:

```
$form.name="operator"

$base.form.name=get.base.form.name($form.name)
```

Example 4:

```
$form.name="operator.a"

$base.form.name=get.base.form.name($form.name)
```

RAD function: get.dateformat

A RAD function that returns the date format, represented by a number, used by your operator ID.

Function
get.dateformat

Format
$date.fmt=get.dateformat()

Factors
Date formats represented, by number:

- 1 = *mm/dd/yy*
- 2 = *dd/mm/yy*

- 3 = *yy/mm/dd*
- 4 = *mm/dd/yyyy*
- 5 = *dd/mm/yyyy*
- 6 = *yyyy/mm/dd*

Example

```
$date.fmt=get.dateformat()
```

After execution for a typical user in the United States, the value of *$date.fmt* is one (*1*).

RAD function: get.graph.action

Description

A RAD function that returns the activated action on a node in the graph diagram.

Function

get.graph.action()

Example

```
$L.graphaction = get.graph.action()
```

RAD function: get.graph.id

Description

A RAD function that returns the value of the graph ID for the expand action on a node in the graph diagram.

Function

get.graph.id()

Example

```
$L.graphid = get.graph.id()
```

RAD function: get.graph.target

Description

A RAD function that returns the target for an activated action on a node in the graph diagram.

Function

get.graph.target()

Example

```
$L.graphtarget = get.graph.target()
```

RAD function: get.graphnode.id

Description

A RAD function that returns the value of the node ID for the expand action on a node in the graph diagram.

Function
get.graphnode.id()

Example
```
$L.graphnodeid = get.graphnode.id()
```

RAD function: get.lock.owner

A RAD function that returns the owner of a resource (lock). The function returns the name of the user who currently has the resource locked.

Function
get.lock.owner

Format
$owner= get.lock.owner("locked resource")

Example
Assume the user "falcon" is actively updating Incident IM1017. By viewing the Lock Resource Status display one can determine that the name of the associated resource is "probsummary;IM1017". To verify the owner from any other application:

```
$lock="probsummary;IM1017"
$owner=get.lock.owner($lock)
```

After execution, the value of $owner is "falcon".

Note: This function is primarily intended for internal use as it requires specific knowledge of the underlying resource names associated with a specific action.

RAD function: get.timezoneoffset

A RAD function that returns the absolute time difference between Greenwich Mean Time (GMT) and the time zone of the operator.

Function
get.timezoneoffset

Format
$tzoffset=get.timezoneoffset()

Example
```
$tzoffset=get.timezoneoffset()
```
After execution for a Pacific Standard Time (PST) time zone user, the value of the *$tzoffset* variable is *-08:00:00*.

RAD function: get.uid

A RAD function that returns a 24-character identifier that is unique to an installation of Service Manager. Each successive call returns a value greater than the previous value.

Function

get.uid

Format

$uid=get.uid()

Example

```
$uid=get.uid()
```

After execution, *$uid* contains a unique value such as 41ed9bf40032329a03039158.

RAD function: gui

A RAD function that determines whether or not a process is running in GUI mode.

Function

gui

Format

$bool=gui()

Example

```
$bool=gui()
```

After execution in GUI mode, the value of *$bool* is *true*. After execution in text mode, value of *$bool* is *false*.

RAD function: index

A RAD function that returns the index or position number for the value of a specific element in an array or a character in a string. If the target value is not in the array or string, NULL is the return.

Function

index

Format

`index(target value, $variable, starting position #)`

Parameters

The `target value` variable is the value you are searching for in the array or string.

The `$variable` variable is the name of the variable to be searched.

The `starting position #` variable is the index in the array or position in the string where the search will begin. The default value is one (*1*).

Factors

- The index() function operates identically for arrays, regardless of the data type in the array.
- If the array is composed of structures or of arrays, the target variable must exactly match the structure or array.
  For example, the function does not allow selecting a field from a structure.
- You can use the index function to search for any value by converting both the value and the search variable to the same type.
  For example, if you want to search for the variable “$problem.number” on all forms, you could pass the following query to the select panel:

`index("$problem.number", str(field))`. The index function searches for the variable "$problem.number" in the 'field' field of the format table.

- When you use the index() function in a query, HP Service Manager evaluates the function based on database case sensitivity.
  For example, if you use the Expert Search option on the Incident Management search screen, and search for `index("Windows Server", brief.description) > 0`:
  - On a case insensitive database, the query returns all records where the brief.description field contains "windows server" or "WINDOWS SERVER" or "Windows SERVER", and so on.
  - On a case sensitive database, the query only returns records where the brief.description field contains exactly "Windows Server".
- When you use the index() function in a RAD or JavaScript expression and not within a query, HP Service Manager evaluates it case sensitively (regardless of the database setting).
  For an example, consider the following RAD or JavaScript expressions:
  - The expression `index("Windows Server", {"WINDOWS SERVER", "windows server"})`, returns 0.
  - The expression `index("Windows Server", {"WINDOWS SERVER", "Windows Server", "windows Server"})`, returns 2.

Example

`index(1,{1,2,3})` returns one (*1*)
`index(2,{1,2,3})` returns two (*2*)
`index(1,{1,2,3},2)` returns zero (*0*)
`index(2,{1,2,3},2)` returns two (*2*)

RAD function: insert

A RAD function that returns an array with one or more inserted elements.

Function

insert

Format

insert($array[, $position[, $number[, $value[, $denull]]]])

Parameters

- The $array parameter is the array into which an element is inserted.
- The $position parameter is the position of the first inserted element. If the value is zero (*0*), the element is inserted at the end of the array. This is an optional parameter. The default value is zero (*0*).
- The $number parameter is the number of elements to insert. This is an optional parameter. The default value is one (*1*). If the value is zero (*0*), this function will return without changing the array.
- The $value parameter is a value to insert into the new elements. This is an optional parameter. The default value is *NULL*.

- The $denull parameter is a logical value that determines whether or not the array should be denulled before inserting elements. This is an optional parameter. The default value is *true*.

Factors

If the source array has NULL elements at the end, this function automatically denulls the array before inserting new elements in any position. To avoid this, set the value of the $denull parameter to *false*.

Examples

The following examples show a desired result and the syntax used to achieve that result.

Action desired	Example and result

Insert an element at the beginning of an array.	`$a={"a", "b", "c", "d"}`
	`$a=insert($a, 1, 1, "z")`
	Where:
	The value of *$a* is *{"z", "a", "b", "c", "d"}*

Insert an element in the middle of an array.	To insert an element in the middle of an array:
	`$a={"a", "b", "c", "d"}`
	`$a=insert($a, 3, 1, "z")`
	Where:
	The value of *$a* is *{"a", "b", "z", "c", "d"}*

Insert an element at the end of an array.	`$a={"a", "b", "c", "d"}`
	`$a=insert($a, 0, 1, "z")`
	Where:
	The value of *$a* is *{"a", "b", "c", "d", "z"}*

The array is denulled.	`$a={"a", "b",  ,  , "e", "f",  ,  ,  , }`
	`$a=insert( $a, 0, 1, "z")`
	Where:
	The value of *$a* is *{"a", "b", , , "e", "f", "z"}*

To avoid denulling the array.	`$a={"a", "b",  ,  , "e", "f",  ,  ,  , }`
	`$a=insert( $a, 0, 1, "z", 0)`
	Where:
	The value of *$a* is *{"a", "b", , , "e", "f", , , , ,"z"}*

The array is (again) denulled.	`$a={"a", "b",  ,  , "e", "f",  ,  ,  , }`
	`$a=insert( $a, 2, 1, "z")`
	Where:
	The value of *$a* is *{"a", "z", "b", , , "e", "f"}*

RAD function: iscurrent

A RAD function that determines if the record that you are working with is identical to the version in the database. A value of true or false is returned.

Function

iscurrent

Format
$Boolean=iscurrent($file.variable)

RAD function: jscall

A RAD function that enables you to invoke JavaScript from within RAD.

Function
jscall

Format
jscall("script.function" , arg1 , arg2, ... , argN)

Parameters
The following parameters are valid for the jscall function:

Parameter	Data type	Required	Description
"script.function"	String	Yes	Refers to a script in the ScriptLibrary and a particular function within the script. This parameter must be enclosed in double quotes.

arg1, arg2, – , argN	Varies with each argument	The required arguments depend upon the requirements of the script you call.	The arguments passed to JavaScript. You can pass an infinite number of arguments. Arguments are not enclosed in quotes or double quotes and can be RAD expressions, as shown in the following examples:
			`city in $L.OLD`
			— OR —
			`level in $L.NEW=0`

RAD function: lioption

This RAD function indicates whether a given input value is a licensed option on the current system.

Function
lioption

Format
lioption(*input value*)

Parameters
Parameters for this function are as follows:

Input value	Description
Asset Contract Management	Authority to use Asset Contract Management application
Change Management	Authority to use Change Management application
Computer Telephony	Authority to use Computer Telephony Interface (CTI)
Configuration Management	Authority to use Configuration Management application
Contract Management	Authority to use Contract Management application
Desktop Administration	Authority to use Desktop Administration application
Incident Management	Authority to use Incident Management application
IR Expert	Authority to use IR Expert
Knowledge Management	Authority to use Knowledge Management application
Knowledge Management ESS	Authority to use Employee Self-Service with the Knowledge Management application
Problem Management	Authority to use Problem Management application
RAD Compiler	Authority to use RAD Compiler
Request Management	Authority to use Request Management application
SCAuto/SDK Unix	Authority to use SC Automate Adapters Unix SDK
SCAuto/SDK Windows	Authority to use SC Automate Adapters Windows SDK
SCAuto/HP Network Node Manager	Authority to use SC Automate Adapters OpenView
SCAuto/Netview (AIX/NT)	Authority to use SC Automate Adapters AIX Netview
SCAuto/SunNet	Authority to use SC Automate Adapters SunNet
SCAuto/SPECTRUM	Authority to use SC Automate Adapters SPECTRUM
SCAuto/HP Operations (ITO/VPO)	Authority to use SC Automate Adapters IT Ops
SCAuto/UNIXmail	Authority to use SC Automate Adapters SMTP or UNIX mail
SCAuto/MAPImail	Authority to use SC Automate Adapters MAPI mail
SCAuto/Lotus Notes	Authority to use SC Automate Adapters Lotus Notes
SCAuto/Fax Unix	Authority to use SC Automate Adapters Unix Fax
SCAuto/Pager Unix	Authority to use SC Automate Adapters Unix Pager

SCAuto/Fax Windows	Authority to use SC Automate Adapters Windows Fax
SCAuto/Pager Windows	Authority to use SC Automate Adapters Windows Pager
SCAuto/Netview OS/390	Authority to use SC Automate Adapters OS/390 Netview
SCAuto/TBSM	Authority to use SC Automate Adapters TBSM
SCAuto/Unicenter AMO	Authority to use SC Automate Adapters Unicenter
SCAuto/Unicenter TNG	Authority to use SC Automate Adapters CA TNG
SCAuto/Tivoli	Authority to use SC Automate Adapters Tivoli
SCAuto/Tally NetCensus	Authority to use SC Automate Adapters Tally NetCensus
SCAuto/Remedy ARS	Authority to use SC Automate Adapters Remedy ARS
SCAuto/ERP Connection	Authority to use SC Automate Adapters ERP Connection
Scheduled Maintenance	Authority to use Scheduled Maintenance application
Self-Service Ticketing	Authority to use Self-Service Ticketing
Service Catalog	Authority to use Service Catalog application
Service Desk	Authority to use Service Desk application
SLA	Authority to use Service Level Agreements application
SOAP API	Authority to publish Service Manager data as Web services

Factors

The function returns a true value if the input value is part of the system license.

The function returns a false value if the input value is not part of the system license.

Examples

The following table describes a desired action and the syntax used to initiate the action.

Action desired	Syntax
Check for authorization to use Change Management	$licensecheck=lioption("Change Management")
Check for authorization to publish Web services	$licensecheck=lioption("SOAP API")

RAD function: lng

A RAD function that returns the number of elements in an array or structure and the number of characters in a string. Remember to denull arrays to get the correct number of elements, but be

careful when denulling arrays of structures.

Function
lng

Format
`lng($variable)`
Where *$variable* is the array, structure, or string.

Factors

- The *$variable* must be an array, structure or a character string.
- Arrays return the number of elements, NULL or otherwise.
- Strings return the number of characters.
- Structures return the number of fields.

Examples
`lng("1234567890")` returns *10*
`lng({1,2,3})` returns *3*
`lng({1,2,3,,,,})` returns *7*
`lng(denull({1,2,3,,,,}))` returns *3*

RAD function: locks

A RAD function that returns an array of structures containing information about the locks used to lock a database for an update or the locks set by RAD applications using a locked command panel.

Function
locks

Format
locks()

Factors
Each element of the array contains:

- lock time
- process id
- terminal id
- operator name
- resource name (name of the lock)
- number
- exclusive
- locked
- breakable

Example
`locks()`
Returns the following:

```
{{['12/26/96 16:01:07', 20212, "SYSTEM", "marquee", "agent:marquee",
0, false, true, false]}
{["01/02/97 11:26:08", 22699, "Windows 32", "falcon", "AG/pm.main",
0, true, true,
false]}}
```

RAD function: logoff

A RAD function that logs off the current user and terminates the session.

Function
logoff

Format
logoff()

Factors
This function does not return a value. It executes a function.

RAD function: logon

A RAD function that logs a user on.

Function
logon

Format
logon()

Factors
This function does not return a value. It executes a function.

RAD function: mandant

The Service Manager mandanten feature protects records of multiple clients by logically separating a shared database. Clients see only the data they are allowed to share. The mandant RAD function allows an application to create a subset of the current mandanten values. For example, if a new ticket is being opened and the client (customer) for the ticket is known, then a subset of the mandanten values can be created to show only the assignments that are valid for that client.

Function
mandant

Format
mandant(n,string)
Where:
The *n* equals 0, 1, 2, or 3

Parameters

- 0 represents a request for a subset of the current mandanten values.
- 1 represents a request to restore the original mandanten values.
- 2 links the current user to the set of records in the `scsecuritygroup` file that matches the array of security IDs.
- 3 sets the security roles for the tables defined in the `scFolderAccess` file.

The value of the *string* variable is the mandanten value.

Example usage

In Service Manager the out-of-box system has three scsecurity group records: "Company A", "Company B" and "Company C". To establish the mandanten security for the user, the RAD could call the following:

```
mandant(2, {"Company A"}) or mandant(2,{"Company A", "Company B"})
```

Security roles are implemented in the scFolderAccess table. The RAD calls the mandant function with 3 to set the security roles for the given tables. The following example sets the security role of ADMIN for two tables, as defined in the `scFolderAccess` file.

```
mandant(3, "ADMIN",{"cm3", "device"})
```

Note: This fails if the string value does not match the values currently established.

RAD function: max

A RAD function that returns the largest value in a list of values or arrays.

Function

max

Format

```
max(element1, element2, element3, ...)
```

— OR —

```
max($variable)
```

Where:

The *$variable* variable is an array or a list of values.

Factors

- The max function returns the maximum value of either times or numbers.
- If you include an array in the list of elements, the system evaluates all of the elements in the array as if they were individual values.
- An attempt to use an array of strings as a parameter results in a segmentation fault, which causes the application to exit in error.

Example

`max(17,24,35,73,10)` returns *73*

`max(1,{2,3,4},3)` returns four (*4*)

RAD function: messages

A RAD function that provides logging functions for use in the memory message log. The log is implemented as a wraparound cache of the *n* most recent messages.

Function
messages

Format
messages($function_number, $function_parameter)

Parameters
Parameters for this function are as follows:

Function #	Parameters	Action
0	0 — 500	Open message log with 0 to 500 entries.
1	None	Start or resume logging.
2	None	Stop logging.
3	None	Stop logging and clear the log.
4	None	Close the log and recover memory.
5	0	Retrieve array of all log entries in most recently displayed order. The log is not cleared.
5	1	Retrieve array of logged informational messages in last displayed order; excludes error and action messages. The log is not cleared.
5	2	Retrieve array of logged action messages in last displayed order; excludes error and informational messages. The log is not cleared.
5	3	Retrieves array of error messages in last displayed order; excludes informational and action messages. The log is not cleared.

Factors
The following table outlines the proper order for each action related to this function and the corresponding function number.

Action	Function
Open log.	Function zero (0)
Start logging.	Function one (1)
...run application	N/A

Stop logging.	Function two (2)
Retrieve log.	Function five (5)
Clear log.	Function three (3)
start logging	Function one (1)
...run application	N/A
...run application	N/A
...run application	N/A
...run application	N/A
Stop logging.	Function two (2)
Retrieve log.	Function five (5)
Close log.	Function four (4)

Notes:

- A user task (the sm process) can have only one private message log at a time.
- You cannot directly access message logs that are using other tasks.
- A maximum of 500 logged messages is supported.

Examples

The following table describes a desired action and the syntax used to initiate the action.

Action desired	Syntax
Open a log for 200 messages.	$throwaway=messages(0,200)
Start logging	$throwaway=messages(1)
Stop logging.	$throwaway=messages(2)
Clear log.	$throwaway=messages(3)
Close log.	$throwaway=messages(4)
Retrieve all logged messages.	$array=messages(5,0)
Retrieve all logged informational messages.	$array=messages(5,1)
Retrieve all logged action messages.	$array=messages(5,2)
Retrieve all logged error messages.	$array=messages(5,3)

RAD function: min

A RAD function that returns the smallest element in a list of values or arrays.

Function

min

Format

`min(element1, element2, element3, ...)`

— OR —

`min($variable)`

Where:

The *$variable* variable is an array or a list of values.

Factors

- The min function returns the minimum value of either times or numbers.
- If you include an array in the list of elements, Service Manager evaluates all elements in the array as individual values.
- An attempt to use an array of strings as a parameter results in a segmentation fault, which will cause the application to exit in error.

Example

`min(17,24,35,73,10)` returns *10*

`min(4,{1,2,3,},2)` returns one (*1*)

RAD function: modtime

A RAD function that returns the last modified date and time of a record.

Function

modtime

Format

modtime($file)

Factors

- Use this function to determine if a record has been modified before or after a specified time.
- Use modtime(currec()) to return the last modified date and time for any record. You can also use this variation in queries to retrieve all records modified after a specified point in time.
- The modtime function is not updated on clients when the records are updated on the server until after the records are selected again.

Examples

`modtime($file)`

Returns *1/10/96 12:15:17*

`modtime(currec)>tod( )- '08:00:00'`

Selects all records in a file that were added or updated in the last eight hours.

RAD function: month

A RAD function that returns the month of year for a specified date, regardless of the date format.

Function
month

Format
$ymonth=month($date)

Parameters
Where:
The *$date* variable is the date and time value

Example
```
$ymonth=month(`2/15/96')
```
After execution, the value of *$ymonth* is two (*2*).

RAD function: multiselect.selection ("fieldcontents")

A RAD function that returns the contents of the field associated with the multiselection. Field contents are only returned if the associated column name is set using the Forms Designer property "Selection Field."

To set the associated column name using the Forms Designer "Selection Field" property:

1. Log on as a System Administrator.
2. Click **Tailoring** > **Forms Designer**.
3. In the Form field, type the form name. For example, type **probsummary.qbe.g** and click **Search**. The form opens.
4. Click **Design** to open design mode.
5. Click on the table object.
6. In the **Selection Field** property, type the name of the column on the table (for example, type **number**).
7. Click **OK** to see what the form will look like in the Windows client.
8. Click **OK** to save your changes.

Function
multiselect.selection("fieldcontents")

Format
$info=multiselect.selection("fieldcontents")

Example
```
$x=multiselect.selection("fieldcontents")
```
After execution, the value of *$x* is set to the selected field contents. For example {"IM10003", "IM10006", "IM10007", "IM10008", "IM10010", "IM10012"}.

Note: Field contents are only returned if the associated column name is set using the Forms Designer property "Selection Field."

RAD function: multiselect.selection("fieldname")

A RAD function that returns the field name of the column associated with the multiselection. The associated column name is set using the Forms Designer property "Selection Field."

To set the associated column name using the Forms Designer "Selection Field" property:

1. Log on as a System Administrator.
2. Click **Tailoring** > **Forms Designer**.
3. In the Form field, type the form name. For example, type **probsummary.qbe.g** and click **Search**. The form opens.
4. Click **Design** to open design mode.
5. Click on the table object.
6. In the **Selection Field** property, type the name of the column on the table (for example, type **number**).
7. Click **OK** to see what the form will look like in the Windows client.
8. Click **OK** to save your changes.

Function
multiselect.selection("fieldname")

Format
$info=multiselect.selection("fieldname")

Example
```
$x=multiselect.selection("fieldname")
```
After execution, the value of *$x* is set to the column name (for example,"number").

Note: The field name is only returned if the associated column name is set using the Forms Designer property "Selection Field."

RAD function: multiselect.selection("rows")

A RAD function that returns an array of row numbers that were selected.

Function
multiselect.selection("rows")

Format
$info=multiselect.selection("rows")

Example
```
$x=multiselect.selection("rows")
```
After execution, the value of *$x* is set to the selected rows. For example {2, 3, 7, 22, 23, 30}.

RAD function: multiselect.selection("selected")

A RAD function that returns true or false if the provided variable has multiple records selected.

Function
multiselect.selection("selected",$file.variable)

Format
$info=multiselect.selection("selected",file.variable)

Example
```
$x=multiselect.selection("selected",$L.file)
```
After execution, the value of *$x* is set to true or false.

RAD function: multiselect.selection("selections")

A RAD function that returns the number of rows selected.

Function
multiselect.selection("selections")

Format
$info=multiselect.selection("selections")

Example
```
$x=multiselect.selection("selections")
```
After execution, the value of *$x* is set to the number of selected rows. For example 6.

RAD function: multiselect.selection("tablename")

A RAD function that returns the Table ID associated with the multiselection.

Function
multiselect.selection("tablename")

Format
$info=multiselect.selection("tablename")

Example
```
$x=multiselect.selection("tablename")
```
After execution, the value of *$x* is set to the selected Table ID. For example, Table0.

RAD function: null

A RAD function that returns a value of true if the parameter is NULL or is comprised of a compound data type consisting of all NULL values. Use the null function to determine whether or not the value of a field or variable is null.

Note: The *null* function is different that the reserved word, *NULL*.

Function

null

Format

null(value)

Factors

- The null function can handle compound data types.
- The reserved word NULL only handles primitive data types.
- An array or structure is null if it contains all null elements. This function applies itself recursively to nested elements.
- An empty string, such as *(" ")* is not null.
- A zero (0) is not null.
- The value *00:00* is not null.
- A value of *unknown* in a Boolean field is not null.

Examples

`null(1)` returns *false*
`null(NULL)` returns *true*
`null({})` returns *true*
`null({1,})` returns *false*
`null({{},})` returns *true*

RAD function: nullsub

A RAD function that substitutes a null value with the second value specified.

Function

nullsub

Format

`nullsub(value1,value2)`

Where:

The *value2* variable is the value to return if *value1* is null.

Examples

`nullsub(1,2)` returns one (*1*).
`nullsub(NULL,2)` returns two (*2*).
`$a=NULL`
`$a=nullsub($a,"test")` returns *test*.

RAD function: operator

A RAD function that sets and returns the name of the current operator ID. The operator function is also used to set the operator name.

Function

operator

Format

operator()

Factors

The *operator()* value is set to the value of the login name in the operator record at the time of login.

Examples

```
$a = operator()
operator() = $a
```

RAD function: option

A RAD function that returns the number of the last interrupt key. For example, 0 = Enter, 1 = first option key, 2 = second option key.

Function

option

Format

option()

Factors

- The Enter key returns a value of zero (0).
- Use caution when option() is a criteria for a decision or process panel that can be accessed from several paths. To avoid conflicts, set a variable to option() when the first panel is accessed.
- Clicking a button returns the number defined in the Button ID property.
- Clicking an item in the More menu returns the GUI option number associated with this option in the displayoption table.

Example

`option()` returns *10*

RAD function: parse

Parses a string into an operator.

Function

parse

Format

parse(string,type)

Factors

- An operator can be an expression, such as *type<11*, or a statement, such as *type=11*.
- When enabled on forms, the parse property automatically parses the data entered.

Examples

```
$x=parse("1+1",1)
```

Evaluate($x) returns two (*2*)

```
$x=parse("$x=1",11)
```

Evaluate($x) causes the value of *$x* to be set to one (*1*).

RAD function: perf

A RAD function that evaluates system performance and then logs this information in Service Manager's systemperform and systemtotals tables. A value of zero (*0*) indicates success, and a value of negative one (*-1*) indicates failure.

Function

perf

Format

`perf(n)`

Where:

The value of *n* is the number that corresponds to the option you select.

Parameters

The following table shows the parameters used with the perf function:

Option	Result
1	Delete records in the systemtotals and systemperform files where capture=false before writing new information to disk.
2	Add records and do not delete. Nothing is removed before new records are written to the systemtotals and systemperform files. Maintenance of these files is typically the responsibility of the RAD programmer.
3	Delete records in the systemtotals and systemperform files where capture=false and do not write information to disk.
4	Delete all records in the systemtotals and systemperform files, regardless of capture status.

Example

`perf(1)` Returns zero (*0*)

RAD function: printer

A RAD function that sets and returns the value of the currently logged-on printer.

Function

printer

Format

printer()

Factors

This function is most often used to set the current printer. When an operator logs on, the login application is executed, assigning printer() to be either the printer specified by the user at login, or the default printer specified in the operator record for the user.

Example

```
$a = printer()
printer() = $a
```

RAD function: policyread

A RAD function that reads the data policy from a datadict table.

Function
policyread

Format
$L.returnvalue = policyread($L.file, fieldname, fieldsetting)

Parameters
The following parameters are valid for the policyread function:

Parameter	Description
$L.returnvalue	The policy value returned after it is read from the database. It can be a string, a boolean, or null.
$L.file	The file handle of the table for which the data policy information is to be read from the database. It can be a join table or a normal table. **Note:** Other kinds of tables (adhoc sql, merge files, etc.) are not supported. This is the value in the **Name** field in the datadict record.
fieldname	The name of the field in the specified table for which the data policy information is to be read from the database. This is the value in the **Field Name** field in the datadict record.
fieldsetting	The field setting on the specified field for which the data policy information is to be read from the database. This one of the fields on **Field Settings** tab of the datadict record. Valid field settings are: "invisible", "readonly", "mandatory", "captions", "avail", "encrypt", "defaults", "globallist", "matchfields", "matchfiles", and "validations".

Factors
If the fieldsetting is not in the list of Field Settings in the datadict table, the server will throw an error message. In any other case, this function returns a valid value or null.

Example 1
This example demonstrates how you could use a JavaScript to access the data policy on a simple file, such as the operator table.

```
// Access DataPolicy definition on operator file.

var operatorFile = new SCFile( "operator" );
var rteReturn;
```

```
rteReturn = system.functions.policyread( operatorFile ,
"profile.change", "invisible" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as
defined in (file:datadict, field:invisible) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile ,
"profile.change", "readonly" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as
defined in (file:datadict, field:readonly) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile ,
"profile.change", "mandatory" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as
defined in (file:datadict, field:mandatory) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile ,
"profile.change", "captions" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as
defined in (file:datadict, field:captions) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile ,
"profile.change", "avail" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as
defined in (file:datadict, field:avail) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile ,
"profile.change", "encrypt" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as
defined in (file:datadict, field:encrypt) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile ,
"profile.change", "defaults" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as
defined in (file:datadict, field:defaults) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile ,
"profile.change", "globallist" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as
defined in (file:datadict, field:globallist) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile ,
"profile.change", "matchfields" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as
defined in (file:datadict, field:matchfields) " + rteReturn );

rteReturn = system.functions.policyread( operatorFile ,
"profile.change", "matchfiles" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as
defined in (file:datadict, field:matchfiles) " + rteReturn );
```

```
rteReturn = system.functions.policyread( operatorFile ,
"profile.change", "validations" );
print( "DataPolicy on (file:operator, fieldname:profile.change), as
defined in (file:datadict, field:validations) " + rteReturn );
```

Example 2
This example demonstrates how you could use a JavaScript to access the data policy on a complex file, such as a join of the incidents and contacts tables.

```
// Access DataPolicy definitions on join file inccontacts (incidents
and contacts)

var incident_contacts_file = new SCFile( "inccontacts" );
var rteReturn;

// Access policy definition on file(contacts) and field(city) and
datadict field defaults
rteReturn = system.functions.policyread( incident_contacts_file,
"file.contacts,city", "defaults" );
print( "DataPolicy on (joinfile:inccontacts, subfile:contacts,
field:city), as defined in(file:datadict, field:defaults)" +
rteReturn );

// Access policy definition on file(incidents) and field(phone) and
datadict field defaults
rteReturn = system.functions.policyread( incident_contacts_file,
"file.incidents,phone", "defaults" );
print( "DataPolicy on (joinfile:inccontacts, subfile:incidents,
field:phone), as defined in(file:datadict, field:defaults)" +
rteReturn );

// Access policy definition on file(incidents) and field(phone), and
datadata field readonly
rteReturn = system.functions.policyread( incident_contacts_file,
"file.incidents,phone", "readonly" );
print( "DataPolicy on (joinfile:inccontacts, subfile:incidents,
field:phone), as defined in(file:datadict, field:readonly)" +
rteReturn );
```

RAD function: processes

A RAD function that returns an array describing all processes.

Function
processes

Format
processes(s)

Factors

Each element of the array contains the following information:

- login time
- session id
- device name
- user name
- idle time
- process id
- thread id
- host name
- ip address

The argument of *s* determines the types of processes to be returned:

S	Type of process
null	All
ALL	All
SYSTEM	System (terminal="SYSTEM")
USER	User (not System)
ACTIVE	Active
INACTIVE	Inactive (not active)

Example

`processes("ACTIVE")` returns *{{['04/06/09 11:02:22', 11, "Soap-Windows Vista", "System.Admin", '00:00:00', 1692, 5164, "HPSD9667", "15.80.163.173"]}}*

RAD function: prof

A RAD function that returns system performance statistics while checking the system resources used by an application.

Function

prof

Format

`prof(n)`

Where:

The value of *n* is the value for the parameter corresponding to the option you want returned.

Parameters

The following parameters are valid for the prof function:

Parameter	Description
1	Returns the user CPU time.
2	This parameter is obsolete.
3	Returns the memory allocated.
4	This parameter is obsolete.
5	Returns the number of statements evaluated.
6	This parameter is obsolete.
7	This parameter is obsolete.
8	This parameter is obsolete.
9	This parameter is obsolete.
10	This parameter is obsolete.
11	This parameter is obsolete.
12	Returns the user executing priority.
13	This parameter is obsolete.
20	Returns the symbol table size.
21	Returns the stack size.

Example

`prof(5)` returns (*7752*).

RAD function: recordcopy

A RAD function that copies a set of fields from one record to another record.

Function

recordcopy

Format

$L.voide=recordcopy($source.file, $source.fields, $target.file, $target.fields)

Parameters

The following parameters are valid for the recordcopy function:

Parameter	Description
$L.void=	Syntax requirement only, does not affect the outcome of the function.
$source.file	The source record from which fields are copied.

$source.fields	Array of field names in $source.file.
$target.file	The target record to which fields are copied.
$target.fields	Array of field names in $target.file.

Factors

The recordcopy function does not check the data type. Ensure that you do not copy a number field to a string, or a structure to a number.

Example

If $source.file is an operator record; $target.file is a contacts record.

```
name in $source.file="falcon"
phone in $source.file="858-481-5000"
$source.fields={"name", "phone"}
$target.fields={"contact.name", "contact.phone"}
$L.void=recordcopy($source.file, $source.fields, $target.file,
$target.fields)
```

After execution, the value of contact.name in $target.file is *falcon* and contact.phone in $target.file is *858-481-5000*.

RAD function: recordtostring

A RAD function that uses the value of a field from an array and appends that value to a string.

Function

recordtostring

Format

recordtostring($string, $file, $arrayofnames, $sep)

Parameters

The following values are valid for the recordtostring function:

Parameter	Description
$string	The string to which the field value is appended.
$file	The current file.
$arrayofnames	The array from which the field is taken.
$sep	A separator character.

Factors

Event Services uses the caret character (^) as a default separator. You can use the recordtostring function to build an eventout string.

Example

```
recordtostring("", $file, {"location", "location.name"}, "^")
```

Where:

The value of *$file* is a location record in which *location=Advantage HQ*, and *location.name=downtown*. After execution, the value of *$string* will contain *Advantage HQ^downtown*.

RAD function: round

A RAD function that rounds up, in the positive direction, to the nearest specified number of decimal places.

Function
round

Format
round(number,n)

Parameters
The following parameters are valid for the round function:

Parameter	Description
number	The number to round.
n	The number of decimal digits to use when rounding. This number can be any input that equates to a numeric value, such as an expression, a variable, or a number.

Example

```
$a = round(3.47,1)
$a = 3.5

$a = round(3.53,1)
$a = 3.5

$a = round(3.45,1)
$a = 3.5

$a = round(-3.45,1)
$a = -3.4

$a = round(100/30,2)
$a = 3.33

$a = round('1/19/96 06:19:34',0)
$a = '1/19/96 06:20'
```

RAD function: same

A RAD function that compares two values. If the values are equal or both values are null, the result is true; otherwise, the result is false.

Function
same

Format

same(value1,value2)

Factors

- Use this function to compare arrays and structures, nulls, or empty strings.
- When using the same function to compare arrays or structures, preface the variable names with denull, as shown in the following example.
  `not same(denull($filea), denull($fileb)).`

Examples

`same(1,NULL)` returns *false*
`same(NULL,NULL)` returns *true*
`same({},{,})` returns *true*
`same({1},{1,})` returns *true*
`same("",NULL)` returns *false*

RAD function: scmsg

Returns a text string matching the format of the message ID number and message class specified. The function replaces any variables in the message with the text specified in an array of values.

Function

scmsg

Format

scmsg($id, $class, $arrayof values)

Parameters

The following parameters are valid for the scmsg function:

Parameter	Data type	Description
$id	Number	The ID number of the message (message.id) from the scmessage table.
$class	String	The Message class from the scmessage table.
$arrayofvalues	Array	An array of substitution text for the *%S* variables in the message text from the scmessage table.

Factors

- All substitution text must be *%S*, which indicates that a string is used to provide the data. Each entry in the array of substitution text is examined for the type. If it is not a string type, it is converted to a string.
- You must create a separate record in the scmessage table for the message in each language you want to display.
- The $arrayofvalues defined in the scmsg function are applied to the message in the scmessage table corresponding to the language defined for the client login.

Example

```
scmsg("21", "cib", {"mail", "middle,caller.id", "open"})
```

Resulting message from the scmessage table:

```
Built macro to %S to %S on %S.
```

When the %S array values from the scmsg function in the example are applied, the message is displayed as:

```
Built macro to mail to middle,caller.id on open
```

RAD function: set.timezone

A RAD function that sets the time zone, the translation from local to internal characters, and the date format. Time zone is established at either user login or process startup time. Note that not all processes that are started are user processes. At login, Service Manager determines the time zone first by the company record and then the operator record.

Function
set.timezone

Format
set.timezone(servertz)

Parameters
The following parameter is valid for the set.timezone function:

Parameter	Data type	Description
servertz	String	A record in a Service Manager tzfile table.

RAD function: setsort

A RAD function that sorts the fields in a table. The setsort function allows for a third parameter so that it could be an array of numbers, zero or one (0 or 1). The entries in this array correspond to the entries in the name array. A "0" indicates the field will be used to sort in ascending order and a "1" indicates the field will be used to sort in descending order. A "0" or "1" can also be used to sort on all fields in ascending or descending order.

Function
setsort

Format

```
setsort($L.file, $L.arrayofnames, 0): ascending sort
setsort($L.file, $L.arrayofnames, 1): descending sort
$L.void=setsort($L.file, $L.arrayofnames, $L.arraysortorder)
```

Example
If you wanted to sort a list of contacts by name (ascending), age (ascending), and state (descending).

```
$L.arrayofnames={"name","age","state"}
$L.arraysortorder={0,0,1}
$L.void=setsort{$L.contacts, $L.arrayofnames, $L.arraysortorder}
```

RAD function: shutdown

A RAD function that shuts down Service Manager from within the system. No values are returned, this function only performs an operation.

Function
shutdown

Format
shutdown()

Factors
Place this function on the last panel to be executed in a shutdown application. Once the function is executed, the form from which Service Manager was started displays.

RAD function: simple.file.load

A RAD function that loads an unload file containing only data records of one file into a target file. If the target file already exists it will either drop the target file first, or return an error based on a flag passed as a parameter.

This function only works with binary unload files. Text unload files are not supported.

WARNING: Due to the limited capabilities of this routine and the potential loss of data, this function should not be used except by system RAD routines such as the upgrade.

Function
simple.file.load()

Format
$L.rc=simple.file.load(<unload file name>, <target file name>, <drop if already exists flag>)

Function returns
true - if successful
false - if there was a failure

Parameters
The following parameters are valid for the simple.file.load function:

Parameter	Data type	Description
unload file name	character	Specifies where the unload file is located
target file name	character	Specifies the target file for this function, for example, signatures. As a safety precaution, this file name has to match the dbdict record internally stored in the unload file, or the function returns an error.
drop if already exists	logical	Controls the behavior of the function if the target file already exists. If set to FALSE, the function will fail and return an error if the target file already exists in the system. If set to TRUE, the function will drop the existing target file first, before recreating it based on the dbdict record stored in the unload file.

Example

```
$L.result=simple.file.load( "C:\\Temp\\my_signatures.unl",
"signatures", false )
```

This example attempts to load the unload file C:\Temp\my_signatures.unl into the signatures file. It will fail if the signatures file already exists (third parameter is false).

Note: The escape characters have to be escaped.

RAD function: str

A RAD function that converts a data type that is not a string into a string.

Function
str

Format
str(*any valid expression*)

Parameters
Any valid expression passed to the str function can be converted to a string.

Examples
`str(1+1)` returns two (*2*)
`"report run at "+ str(tod())` returns *"report run at 12/10/90 10:10:00"*

RAD function: stradj

A RAD function that ensures a string is a specific length by removing or adding trailing blanks.

Function
stradj

Format
$L.void=stradj($string, $size)

Parameters
The following parameters are valid for the stradj function:

Parameter	Data type	Description
$L.void=	Not applicable	A syntax requirement only, which does not affect the outcome of the function.
$string	String	The string being modified.
$size	Number	The desired size of $string.

Examples

```
$string="Hewlett-Packard"
$size=15
$L.void=stradj($string, $size)
```

After execution, the value of *$string* is *Hewlett-Packard*.

```
$string="Hewlett-Packard"
$size=10
$L.void=stradj($string, $size)
```

After execution, the value of *$string* is *Hewlett-Pa*.

RAD function: strchrcp

A RAD function that replaces part of a string with copies of another string.

Function
strchrcp

Format
$L.void=strchrcp($target, $index, $source, $copies)

Parameters
The following values are valid for the strchrcp function:

Parameter	Data type	Description
$L.void=	Not applicable.	A syntax requirement only, does not affect the outcome of the function.
$target	String	The string being modified.
$index	Number	The position in *$target* to begin copying *$source*.
$source	String	The string to be copied into *$target*.
$copies	Number	The number of copies of *$source* to replicate within *$target*.

Factors
This function never extends the length of $target.

Example

```
$target="Hewlett-Packard"
$index=4
$source="falcon"
$copies=2
$L.void=strchrcp($target, $index, $source, $copies)
```

After execution, the value of *$target* is *Hewfalconfalcon*.

RAD function: strchrin

A RAD function that inserts copies of a string into another string.

Function
strchrin

Format
$L.void=strchrin($target, $index, $source, $copies)

Parameters

The following parameters are valid for the strchrin function:

Parameter	Data type	Description
$L.void=	Not applicable.	A syntax requirement only, does not affect the outcome of the function.
$target	String	The string being modified.
$index	Number	The position in *$target* to begin inserting copies of *$source*.
$source	String	The string to be inserted into *$target*.
$copies	Number	The number of copies of *$source* to insert into *$target*.

Example

```
$target="Hewlett-Packard"
$index=4
$source="falcon"
$copies=2
$L.void=strchrin($target, $index, $source, $copies)
```

After execution, the value of *$target* is *Hewfalconfalconlett-Packard*.

RAD function: strclpl

A RAD function that removes a number of characters from the beginning of a string.

Function

strclpl

Format

$L.void=strclpl($string, $number)

Parameters

The following parameters are valid for the strclpl function:

Parameter	Data type	Description
$L.void=	Not applicable.	A syntax requirement only, does not affect the outcome of the function.
$string	String	The string being modified.
$number	Number	The number of characters to remove.

Example

```
$string="Hewlett-Packard"
$number=6
$L.void=strclpl($string, $number)
```

After execution, the value of *$string* is *t-Packard*.

RAD function: strclpr

A RAD function that removes a specified number of characters from the end of a string.

Function
strclpr

Format
$L.void=strclpr($string, $number)

Parameters
The following parameters are valid for the strclpr function:

Parameter	Data type	Description
$L.void=	Not applicable.	A syntax requirement only. It does not affect the outcome of the function.
$string	String	The string being modified.
$number	Number	The number of characters to remove.

Example

```
$string="Hewlett-Packard"
$number=6
$L.void=strclpr($string, $number)
```

After execution, the value of *$string* is *Hewlett-P*.

RAD function: strcpy

A RAD function that replaces a part of a string with a substring.

Function
strcpy

Format
$L.void=strcpy($target, $tindex, $source, $sindex, $number)

Parameters
The following parameters are valid for the strcpy function:

Parameter	Data type	Description
$L.void=	Not applicable.	A syntax requirement only, does not affect the outcome of the function.
$target	String	The string being modified.
$tindex	Number	The position in *$target* to begin copying a substring from *$source*.
$source	String	The string supplying a substring.

$sindex	String	The beginning position of the substring within *$source*.
$number	Number	The number of characters to copy from *$source*.

Example

```
$target="Hewlett-Packard"
$tindex=4
$source="falcon"
$sindex=2
$number=3
$L.void=strcpy($target, $tindex, $source, $sindex, $number)
```

After execution, value of *$target* is *Hewalct-Packard*.

RAD function: strdel

A RAD function that deletes a number of characters from a specific location in a string.

Function
strdel

Format
$L.void=strdel($string, $index, $number)

Parameters
The following parameters are valid for the strdel function:

Parameter	Data type	Description
$L.void=	Not applicable.	A syntax requirement only, does not affect the outcome of the function.
$string	String	The string being modified.
$index	Number	The location in *$string* from which characters are to be deleted.
$number	Number	The number of characters to delete from *$string*.

Example

```
$string="Hewlett-Packard"
$index=6
$number=8
$L.void=strdel($string, $index, $number)
```

After execution, the value of *$string* is *Hewletd*.

RAD function: strins

A RAD function that inserts a substring into another string.

Function
strins

Format

$L.void=strins($target, $tindex, $source, $sindex, $number)

Parameters

The following parameters are valid for the strins function:

Parameter	Data type	Description
$L.void=	Not applicable.	A syntax requirement only, does not affect the outcome of the function.
$target	String	The string being modified.
$tindex	Number	The position in *$target* to begin inserting a substring from *$source*.
$source	String	The string supplying a substring.
$sindex	String	The beginning position of the substring within *$source*.
$number	Number	The number of characters to insert from *$source*.

Example

```
$target="Hewlett-Packard"
$tindex=4
$source="falcon"
$sindex=2
$number=3
$L.void=strins($target, $tindex, $source, $sindex, $number)
```

After execution, value of *$target* is *Hewalclett-Packard*.

RAD function: strpadl

A RAD function that pads a string with leading blanks until the string is a specified size.

Function

strpadl

Format

$L.void=strpadl($string, $size)

Parameters

The following parameters are valid for the strpadl function:

Parameter	Data type	Description
$L.void=	Not applicable.	A syntax requirement only, does not affect the outcome of the function.
$string	String	The string being modified.
$size	Number	The desired size of *$string*.

Example

```
$string="Hewlett-Packard"
```

```
$size=20
$L.void=strpadl($string, $size)
```

After execution, value of *$string* is *(space)(space)(space)(space)(space)Hewlett-Packard*.

RAD function: strpadr

A RAD function that pads a string with trailing blanks until the string is a specified size. If the string is longer than the size desired, it is truncated.

Function

strpadr

Format

$L.void=strpadr($string, $size)

Parameters

The following parameters are valid for the strpadr function:

Parameter	Data type	Description
$L.void=	Not applicable.	A syntax requirement only. It does not affect the outcome of the function.
$string	String	The string being modified.
$size	Number	The desired size of string.

Examples

```
$string="Hewlett-Packard"
$size=20
$L.void=strpadr($string, $size)
```

After execution, the value of *$string* is *Hewlett-Packard(space)(space)(space)(space)(space)*.

```
$string="Hewlett-Packard"
$size=10
$L.void=strpadr($string, $size)
```

After execution, the value of *$string* is *Hewlett-Pa*.

RAD function: strrep

A RAD function that returns a string after replacing a specified character sequence with another string.

Function

strrep

Format

strrep($target,$string1,$string2)

Parameters

The following parameters are valid for the strrep function:

Parameter	Data type	Description
$target	String	The string containing the characters specified to be replaced.
$string1	String	The string within *$target* to be replaced.
$string2	String	The string replacing *$string1*.

Example

```
$s=strrep("Personnel","sonnel","formance")
```

After execution, the value of *$s* is *Performance*.

RAD function: strtrml

A RAD function that removes all leading blanks from a string.

Function
strtrml

Format
$L.void=strtrml($string)

Parameters
The following parameters are valid for the strtrml function:

Parameter	Data type	Description
$L.void=	Not applicable.	A syntax requirement only. It does not affect the outcome of the function.
$string	String	The string being modified.

Note: This RAD function is not available as a JavaScript system.functions call.

Example

```
$string="  Hewlett-Packard"
$L.void=strtrml($string)
```

After execution, the value of $*string* is *Hewlett-Packard*.

RAD function: strtrmr

A RAD function that removes all trailing blanks from a string.

Function
strtrmr

Format
$L.void=strtrmr($string)

Parameters
The following parameters are valid for the strtrmr function:

Parameter	Data type	Description
$L.void=	Not applicable.	A syntax requirement only, does not affect the outcome of the function.
$string	String	The string being modified.

Note: This RAD function is not available as a JavaScript system.functions call.

Example

```
$string="Hewlett-Packard  "
$L.void=strtrmr($string)
```

After execution, the value of *$string* is *Hewlett-Packard*.

RAD function: substr

A RAD function that extracts a substring from a string.

Function
substr

Format
substr(string, beginning position, length)

Parameters
The following parameters are valid for the substr function:

Parameter	Data type	Description
$string	String	The string being modified.
beginning position	Number	The position in the string where the substring is to begin.
length	Number	The total number of characters you want the substring to contain. The default behavior is to include all remaining characters in the string from the starting position. You can use the length parameter to specify a shorter substr length.

Example

```
$a=substr(" Service Manager", 8)
$b=substr(" Service Manager", 1, 7)
```

After execution, the value of *$a* is *Manager* and the value of *$b* is *Service*.

RAD function: substrb

A RAD function that extracts a substring from a multibyte character or binary string.

Function

substrb

Format

substrb(string, beginning position, length)

Parameters

The following parameters are valid for the substr function:

Parameter	Data type	Description
string	String	The string being modified.
beginning position	Number	The position in the string where the substring is to begin.
length	Number	The total number of bytes you want the substring to contain. The default behavior is to include all remaining bytes in the string from the starting position. You can use the length parameter to specify a shorter byte length. **Important**: You should choose a length value that takes into account the number of bytes available in the source string. If you choose too low a value the substituted string may contain corrupted data because the function did not have enough bytes to substitute a multibyte character. Refer to a UTF-8 or UTF-16 character encoding map to see how many bytes a particular multibyte character requires. It is recommended that you only use this function on strings of a fixed byte length to avoid possible data corruption. If you want to replace a fixed number of characters use the **substr** function instead.

Example

```
$a=substrb("abcÄÖÜdef", 5)
$b=substrb("abcÄÖÜdef", 1, 7)
```

After execution, the value of *$a* is *ÖÜdef* and the value of *$b* is *abcÄÖ*.

For the string $b, setting the length value to 4 or 6 would lead to data corruption because there are not enough bytes available to contain the multibyte characters Ä and Ö respectively.

RAD function: sysinfo.get("ActiveFloatUsers")

A RAD function that returns the total number of floating users currently active.

Function

sysinfo.get("ActiveFloatUsers")

Format

$info=sysinfo.get("ActiveFloatUsers")

RAD function: sysinfo.get("ActiveLicenses")

A RAD function that returns the total number of licenses currently active.

Function
sysinfo.get("ActiveLicenses")

Format
$info=sysinfo.get("ActiveLicenses")

RAD function: sysinfo.get("ActiveNamedUsers")

A RAD function that returns the total number of named users currently active.

Function
sysinfo.get("ActiveNamedUsers")

Format
$info=sysinfo.get("ActiveNamedUsers")

RAD function: sysinfo.get("AuthMode")

A RAD function that returns the authentication mode of the current user:

- Returns *db* if using password authentication and the password exits in database.
- Returns *ldap* if using password authentication and the password exits in LDAP.
- Returns *tso* if using TSO.
- Returns *lwsso* if using LWSSO.

Function
sysinfo.get("AuthMode")

Format
$info=sysinfo.get("AuthMode")

Example
```
$info=sysinfo.get("AuthMode")
```
After execution, the value of *$info* is set to the authentication mode of the current user.

RAD function: sysinfo.get("ClientNetAddress")

A RAD function that returns the Service Manager client network IP address as a character string.

Function
sysinfo.get("ClientNetAddress")

Format
$info=sysinfo.get("ClientNetAddress")

Example

```
$x.clientNetAddress=sysinfo.get("ClientNetAddress")
```

RAD function: sysinfo.get("ClientOSName")

A RAD function that returns the name of the operating system (OS) that Service Manager is running on, such as Windows XP or Mac OS.

Function
sysinfo.get("ClientOSName")

Format
$info=sysinfo.get("ClientOSName")

RAD function: sysinfo.get("ClientPID")

A RAD function that returns the Process ID (PID) for RAD. The client PID is the same as the server PID.

Function
sysinfo.get("ClientPID")

Format
$info=sysinfo.get("ClientPID")

Example

```
$x.clientPid=sysinfo.get("ClientPID")
```

RAD function: sysinfo.get("ClientVersion")

A RAD function that returns the version number of the client software as a string in the form of *release.version.level*, for example, *7.0.1*.

Function
sysinfo.get("ClientVersion")

Format
$info=sysinfo.get("ClientVersion")

RAD function: sysinfo.get("Display")

A RAD function that returns the following:

- Returns *gui* if the user is running a GUI client.
- Returns *text* if the process is not running through a GUI client, such as a background process.

Function
sysinfo.get("Display")

Format
$info=sysinfo.get("Display")

Example

```
$display=sysinfo.get("display")
```

After execution, the *$display* variable is *gui* if the user is running a Windows session.

```
If sysinfo.get("Display")="gui" then (...)
If index(sysinfo.get("Display"), "gui")>0 then (...)
If sysinfo.get("Display")~="text" then (...)
```

RAD function: sysinfo.get("Environment")

A RAD function that returns the following:

- Returns *scserver* if your session is a client application connection such as ConnectIT or GetServices.
- Returns *sm* if your session is a background process such as a scheduler.
- Returns *scguiwswt* if your session is a Windows client connection.
- Returns *scguiwweb* if your session is a Web client connection.

Function
sysinfo.get("Environment")

Format
$info=sysinfo.get("Environment")

RAD function: sysinfo.get("languagecode")

A RAD function that returns the server session language code value used for localized forms and messages.

Function
sysinfo.get("languagecode")

Format
$info=sysinfo.get("languagecode")
Note: If no language has been set, "en" is returned.

RAD function: sysinfo.get("MaxFloatUsers")

A RAD function that returns the total number of floating users logged on since startup.

Function
sysinfo.get("MaxFloatUsers")

Format
$info=sysinfo.get("MaxFloatUsers")

RAD function: sysinfo.get("MaxLicenses")

A RAD function that returns the maximum number of licenses used since startup.

Function
sysinfo.get("MaxLicenses")

Format
$info=sysinfo.get("MaxLicenses")

RAD function: sysinfo.get("Mode")

A RAD function that returns the following:

- Returns *server* if RAD is running in server mode.
- Returns *sm* if RAD is running on a sm process.
- Returns *SOAP* if RAD is running on a Windows or Web client.
- Returns *express* if RAD is running in express mode.

Function
sysinfo.get("Mode")

Format
$info=sysinfo.get("Mode")

Examples

```
$mode=sysinfo.get("mode")
```

Returns *SOAP* if the user is running on a Windows or Web client.

```
If sysinfo.get("mode")="client" then (...)
```

```
If (index(sysinfo.get("mode"),{"sm","server",
"express"})>0 then (...)
```

RAD function: sysinfo.get("PrevLabel")

A RAD function that returns the label name of the last RAD panel executed, thus helping determine where errors occur. A RAD program may also require the label of the last RAD panel that was executed.

Function
sysinfo.get("PrevLabel")

Format
$info=sysinfo.get("PrevLabel")

Example

```
$L.lastlabel = sysinfo.get("prevlabel")
```

The panel name is the last panel executed in the RAD program called. At the start panel for any application, the name is the *calling* panel name from the parent application.

RAD function: sysinfo.get("PrintOption")

A RAD function that returns printing option information. The return is either *client* or *server*, depending on how the system is set.

Function
sysinfo.get("PrintOption")

Format
$info=sysinfo.get("PrintOption")

RAD function: sysinfo.get("Quiesce")

A RAD function that returns *true* if the system is quiesced and *false* if the system is not quiesced. When the system is quiesced, it is readonly.

Function
sysinfo.get("Quiesce")

Format
$info=sysinfo.get("Quiesce")

RAD function: sysinfo.get("RecList")

A RAD function that returns *true* if record list is enabled and *false* if record list is disabled.

Function
sysinfo.get("RecList")

Format
$info=sysinfo.get("RecList")

RAD function: sysinfo.get("ServerNetAddress")

A RAD function that returns the Service Manager server network IP address in the form of a character string.

Function
sysinfo.get("ServerNetAddress")

Format
$info=sysinfo.get("ServerNetAddress")

RAD function: sysinfo.get("ServerNetPort")

A RAD function that returns the Service Manager server port used by the connected client or the system name when called by a background process.

Function

sysinfo.get("ServerNetPort")

Format

$info=sysinfo.get("ServerNetPort")

Factors

Returns either the port number or the service name, as specified .

Examples

The system started with:

```
sm -httpPort:12932
```

A connected client issuing this RAD function returns 12932

The system started with:

```
sm --httpPort:scport1
```

A connected client issuing this RAD function returns scport1.

RAD function: sysinfo.get("ServerPID")

A RAD function that returns the Process ID (PID) of the server, in the form of a number.

Function

sysinfo.get("ServerPID")

Format

$info=sysinfo.get("ServerPID")

RAD function: sysinfo.get("Telephony")

A RAD function that returns *true* if telephony is enabled and *false* if telephony is disabled.

Function

sysinfo.get("Telephony")

Format

$info=sysinfo.get("Telephony")

RAD function: sysinfo.get("ThreadID")

A RAD function that returns the number of the current RAD thread.

Function

sysinfo.get("ThreadID")

Format

$info=sysinfo.get("ThreadID")

Example

```
$x=sysinfo.get("ThreadID")
```

After execution, the value of *$x* is set to the number ID of the current RAD thread.

RAD function: sysinfo.get("TotalFloatUsers")

A RAD function that returns the total number of floating users allowed.

Function
sysinfo.get("TotalFloatUsers")

Format
$info=sysinfo.get("TotalFloatUsers")

RAD function: sysinfo.get("TotalLicenses")

A RAD function that returns the total number of licenses allowed.

Function
sysinfo.get("TotalLicenses")

Format
$info=sysinfo.get("TotalLicenses")

RAD function: sysinfo.get("TotalNamedUsers")

A RAD function that returns the total number of named users allowed.

Function
sysinfo.get("TotalNamedUsers")

Format
$info=sysinfo.get("TotalNamedUsers")

RAD function: sysinfo.get("TotalProcs")

A RAD function that returns the total number of executing processes.

Function
sysinfo.get("TotalProcs")

Format
$info=sysinfo.get("TotalProcs")

RAD function: sysinfo.get("TotalSystemProcs")

A RAD function that returns the total number of system processes.

Function
sysinfo.get("TotalSystemProcs")

Format
$info=sysinfo.get("TotalSystemProcs")

RAD function: sysinfo.get("TotalUserProcs")

A RAD function that returns the total user processes.

Function
sysinfo.get("TotalUserProcs")

Format
$info=sysinfo.get("TotalUserProcs")

RAD function: time

A RAD function that returns the time portion of a date and time value.

Function
time

Format
time($date.time.variable)

Parameters
The following parameter is valid for the time function:

Parameter	Data type	Description
$string	String	The string variable containing upper-case characters.

Factors
The time of a given date and time may be different in different time zones.

Example
```
time($date.time)
```
After execution, the value of *$date.time* is *12:10:15*.

RAD function: tod

A RAD function that returns the current system date and time in the logged on user's timezone and date format.

Function
tod

Format
tod()

Factors
The tod function returns *mm*/*dd*/*yyhh*:*mm*:*ss*.

Examples
```
tod()=`4/20/96 12:15:16'
$curr.time=tod()
```

RAD function: tolower

A RAD function that converts all uppercase characters in a string variable to lowercase characters. Characters with no lowercase equivalent remain the same.

Function
tolower

Format
tolower($string)

Parameters
The following parameter is valid for the tolower function:

Parameter	Data type	Description
$string	String	The string variable containing uppercase characters.

Factors
Character conversions are based on the value of the language parameter.

Example

```
$s=tolower("Copyright 2011")
```

After execution, the value of *$s* is *copyright 2011*.

RAD function: toupper

A RAD function that converts all lower–case characters in a string variable to upper–case characters. Characters with no upper–case equivalent remain the same.

Function
toupper

Format
toupper($string)

Parameters
The following parameter is valid for the toupper function:

Parameter	Data type	Description
$string	String	The string variable containing lower-case characters.

Factors
Character conversions are based on the value of the language parameter.

Example

```
$s=toupper("Copyright 2004")
```

After execution, the value of *$s* is *COPYRIGHT 2004*.

RAD function: translate

A RAD function that translates from one character set to another.

Programming considerations: This function has been deprecated and is maintained for backward compatibility only. This function can only be used on simple single byte character strings and is not UTF-8 safe. Case conversion on strings should be done with the **toupper()** and **tolower()** functions, which are UTF-8 safe.

Function
translate

Format
translate(source, old character set, new character set)

Parameters
The following parameters are valid for the translate function:

Parameter	Data type	Description
source	String	The string to translate.
old character set	String	The existing character set.
new character set	String	The character set to translate to.

Factors

- Old and new character sets must be strings of the same length.
- Old and new character sets may be literals, variables, or fields.
- A one–for–one correlation must exist between the old and new character sets. For example, the third character in the new set replaces the third character in the old set.

Example

```
translate("123abcdef","abc","ABC")
```

After execution, the value is *123ABCdef*.

RAD function: trunc

A RAD function that truncates the decimal digits of a number to a specified number of digits.

Function
trunc

Format
trunc(number,n)

Parameters

Parameter	Data type	Description
number	Number	The number to truncate.
n	Number	The number of decimal digits desired. The default value of *n* is zero (0).

Factors

- The second parameter is not required. By default, the number passed is truncated to a whole

number.

- Negative and positive numbers are treated the same.
- Character conversions are based on the value of the language parameter.

Examples

`trunc(3.45,1)`
After execution, *3.4* is returned.
`trunc(3.44)`
After execution, *3* is returned.
`trunc('13:19:34')`
After execution, *13:19:00* is returned.

RAD function: type

A RAD function returns the numeric type of a datum. See "Data types" on page 20 for a numeric representation of each data type.

Function
type

Format
$ttype=type($any.value)

Parameters
The following parameter is valid for the type function:

Parameter	Data type	Description
$any.value	String	Any literal or variable.

Example

`$ttype=type('2/15/96')`
After execution, the value of *$ttype* is three (*3*).

RAD function: val

A RAD function that converts a datum to a different data type.

Function
val

Format
$new.data=val($data, $new.type)

Parameters
The following parameters are valid for the val function:

Parameter	Data type	Description
$data	Any	Datum to be converted.

	data type	
$new.type	Number	Data type to convert the data to. The default value is one (*1*). For a list of the numbers that correspond to the data types, see "Data types" on page 20.

Factors

- A number can be converted to date and time, a string, or logical.
- A string can be converted to a number, date and time, or logical.
- A date and time can be converted to a number or a string.
- Logical can be converted to a number or a string.

Examples

The following examples show the value after execution:

Variable	Value of *$x* after execution
$x=val(100)	100
$x=val(100, 2)	"100"
$x=val(100, 3)	‘00:01:40’
$x=val(0, 4)	false
$x=val(1, 4)	true
$x=val(2, 4)	unknown
$x=val(3, 4)	NULL
$x=val("HP", 1)	NULL
$x=val("100.3", 1)	100.3
$x=val("HP", 2)	"HP"
$x=val("HP", 3)	NULL
$x=val("2/15/96", 3)	‘02/15/96 00:00:00’
$x=val("true", 4)	true
$x=val("0", 4)	NULL
$x=val(‘01:00:23’, 1)	3623
$x=val(‘02/16/96 08:00:00’)	62992915200
$x=val(‘01/01/96’, 2)	“01/01/96 00:00:00”
$x=val(‘02/09/96 08:00:00’, 3)	‘02/09/96 08:00:00’

$x=val('01/01/96', 4)	NULL
$x=val('12:34:56', 4)	NULL
$x=val(false)	0
$x=val(true, 1)	1
$x=val(unknown, 1)	2
$x=val(false, 2)	"false"
b	NULL
$x=val(unknown, 4)	unknown

RAD function: version

A RAD function that returns an array containing the release number of the Service Manager executable.

Function
version

Format
version()

Factors
Character conversions are based on the value of the language parameter.

Example
```
$a=version()
```
After execution, the value of *$a* is *{"unix",7.0,"7.0.1"}*.

- The first element is a string naming the platform: unix, mswin, or winnt.
- The second element is *7.0*, the RAD release level.
- The third element is a string containing the Service Manager version number and the Service Pack number, such as *7.0.1* indicates *Service Manager version 7.0, Service Pack 1*.

RAD function: year

A RAD function that returns the full year for a date, regardless of the date format.

Function
year

Format
$fyear=year($date)

Parameters

The following parameter is valid for the year function:

Parameter	Data type	Description
$date	Date/time	The date and time value.

Example

```
$fyear=year('2/15/05')
```

After execution, the value of $*fyear* is *2005*.

List: rtecalls()

The rtecalls are based on one RAD function, the rtecall( ) function. They are in-line statements that are useful in the Service Manager RAD Debugger. The rtecall function is available anywhere in Service Manager that supports expressions, statements, or calculations. For example, rtecalls can be used in Format Control calculations, scripts, and displayoptions. The rtecall() function is extensible. New capabilities may appear in every release of Service Manager.

Function	Description
rtecall ("alalnum")	Checks to make sure a string contains only alphanumeric characters, or only alphanumeric characters and the provided non-alphanumeric characters.
rtecall ("counter")	Turns counters on or off for the current session. Other Service Manager users are unaffected.
rtecall ("datemake")	Returns a date, in the proper form, based upon a series of numbers passed to it.
rtecall("escstr")	Precedes special characters in a string with an escape character.
rtecall("filldate")	Places the current date and time in a field in the current record.
rtecall ("filecopy")	Copies all of the data in a collection to another file variable.
rtecall("fileinit")	Initializes a new file (rinit) in $targetfile.
rtecall ("getnumber")	Replaces the getnumb RAD application.
rtecall ("getrecord")	Retrieves the record identified by Unique key values in $L.array.
rtecall ("getunique")	Returns into $L.array the values for the Unique key from the current record in $L.file.
rtecall ("isalnum")	Checks to make sure a string contains only numeric characters, or only numeric characters and the provided non-numeric characters.
rtecall ("isalpha")	Checks to make sure a string contains only alphabetic characters, or only alphabetic characters and the provided non-alphabetic characters.

rtecall ("islower")	Checks to make sure a string contains only lowercase characters, or only lowercase characters and the provided non-lowercase characters.
rtecall ("isnumeric")	Checks to make sure a string contains only numeric characters, or only numeric characters and the provided non-numeric characters.
rtecall ("isupper")	Checks to make sure a string contains only uppercase characters, or only uppercase characters and the provided non-uppercase characters.
rtecall("log")	Sends a message to the external sm.log file.
rtecall ("passchange")	Changes this user's password.
rtecall ("policycheck")	Imposes data policy as defined in the datadict table.
rtecall ("qbeform")	Returns a record list form, which can be passed into an rio or fdisp panel.
rtecall ("recdupl")	Copies the contents of the current record into the contents of another record.
rtecall ("recordexists")	Specifies whether any records exist that match a particular query regardless of any Mandanten data restrictions.
rtecall("refresh")	Retrieves the most current value of a record from the database and returns it in the file variable.
rtecall("rfirst")	Places the pointer at the first record in a record collection (a record list).
rtecall("rgoto")	Places the pointer at the indicated record.id in a record collection (a record list).
rtecall("rid")	Returns the record number of the current record (represented by $L.file).
rtecall ("scantable")	Scans the specified table for certain types of invalid records and fixes them.
rtecall("sort")	Sorts a list or a list of lists in ascending or descending order.
rtecall ("statusupdate")	Publishes a message to a specific field on a form.
rtecall ("transtart")	Measures the amount of data transferred, elapsed time and CPU usage of any transaction.
rtecall ("transtop")	Measures the amount of data transferred, elapsed time and CPU usage of any transaction. Transtop is commonly invoked from the RAD debugger and is used in conjunction with transtart.
rtecall("trigger")	Turns triggers on or off for the current session.

rtecall("alalnum") function

A RAD function that verifies that a string contains only alphanumeric characters. The first character must be alphabetic. The remaining characters must be alphabetic, numeric, or one of the non-alphabetic characters provided.

Function

rtecall("alalnum")

Format

$L.success.flag= rtecall ($L.fnc.name, $L.return.code, $L.str, $L.chars)

Parameters

The following parameters are valid for the rtecall("alalnum") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates if the function was successful
$L.fnc.name	String	Name of the sub-function to call, in this case "alalnum"
$L.return.code	Number	Provides a more detailed return code
$L.str	String	The string to be checked
$L.chars	String	An optional comma delimited string of non-alphanumeric characters to allow

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.success.flag=true
$L.return.code=0
$L.str=name in $L.file
$L.chars=".,_"
$L.success.flag=rtecall("alalnum", $L.return.code, $L.str, $L.chars)
```

Results:

- $L.success.flag is true when name in $L.file="my.name"
- $L.success.flag is true when name in $L.file="my_name"
- $L.success.flag is false when name in $L.file="my name"

rtecall("counter") function

A RAD function that turns counters on or off for the current session. Other users are not affected.

Function

rtecall("counter")

Format

$L.success.flag= rtecall($L.fnc.name, $L.return.code, $L.switch)

Parameters

The following parameters are valid for the rtecall("counter") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates whether or not the function was successful.
$L.fnc.name	String	The name of the sub-function to call, in this case "counter".
$L.return.code	Number	Provides a more detailed return code.
$L.switch	Number	The value is either one (*1*), indicating counters on, or zero (*0*), indicating counters off.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
 $L.success.flg=rtecall("counter", $L.return.code, 1) to turn on
counters
 $L.success.flg=rtecall("counter", $L.return.code, 0) to turn off
counters
```

rtecall("datemake") function

A RAD function that returns a date, in the proper form, based upon a series of numbers passed to it.

Function

rtecall("datemake")

Format

$L.success.flg=rtecall($L.fnc.name, $L.return.code, $L.date, $L.yr, $L.mo, $L.da, $L.hr, $L.mn, $L.se)

Parameters

The following parameters are valid for the rtecall("datemake") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates if the function was successful.
$L.fnc.name	String	The name of the sub-function to call, in this case "datemake".
$L.return.code	Number	Provides a more detailed return code.
$L.date	Date/Time	The variable in which the date will be returned.
$L.yr	Number	The year. A 2-digit year uses the prefix *20* for years up to 50, and *19* for years after. For example, *48* returns *2048*, *99* returns *1999*.

$L.mo	Number	The month.
$L.da	Number	The day. Valid values are from one (1) through 31.
$L.hr	Number	The hour.
$L.mn	Number	The minutes.
$L.se	Number	The seconds.

Factors

The $L.success.flag and $L.return.code always return *true*, even when the function is unsuccessful.

Examples

```
$L.cal.date='01/01/05 00:00:00'
$L.success.flg=rtecall("datemake",$L.return.code, $L.cal.date, 0, 5,
31, 17, 15, 22)
```

Results:

- $L.success.flag=*true*
- $L.cal.date='05/31/2005 17:15:22'

```
$L.cal.date='01/01/05 00:00:00'
$L.success.flag= rtecall("datemake",$L.return.code, $L.cal.date, 0,
2, 31, 17, 15,
22)
```

Note that *2/31/05* is an invalid date.

Results:

- $L.success.flag=true
- $L.cal.date='01/01/2005 00:00:00'

The result is invalid; therefore the date variable is unchanged. Regardless of the success or failure, the $L.success.flag returns a value of *true*.

rtecall("escstr") function

A RAD function that precedes special characters in a string with an escape character. This function is rarely needed. The only situation in which a string needs the escape characters added is when that string is to be fed back into Service Manager. Service Manager treats any data between quotes as a string. If the data contains a quote, then that quote must be *escaped* with a backslash escape character (\) to ensure that the quote is treated as data instead of the end of the string. Since the backslash is used as the escape character, any occurrence of a backslash in the data must also be *escaped*.

An example of when this function might be needed is when a RAD program is constructing a query to retrieve a record based on data from another record. In this case, the contact name in an incident retrieves the contact information from the contacts table. The RAD program might construct a query as follows:

```
$L.query = "name="+contact.name in $file
```

This query will not function properly if the *contact.name* from $file contains a quote or a backslash character because these characters are not properly *escaped* and will cause the parse of the query to end prematurely. The correct example is as follows:

```
$L.temp = contact.name in $file
$L.ret = rtecall("escstr",$L.rc,$L.temp)
$L.query="name="+$L.temp
```

Function

rtecall("escstr")

Format

rtecall($L.fnc.name, $L.rc, $L.str)

Parameters

The following parameters are valid for the rtecall("escstr") function:

Parameter	Data type	Description
$L.fnc.name	String	Name of the sub-function to call, in this case "escstr".
$L.rc	Number	Provides a more detailed return code.
$L.str	String	The string to be modified.

Example

Before the call to *"escstr"*, *$L.str* contained **c:\dir\sub**

```
$L.rc=0
$L.ret=rtecall("escstr", $L.rc, $L.str)
```

After the rtecall, *$L.str* contains the following string: **c:\\dir\\sub**. The backslash escape character (\) is now inserted in front of the existing backslash.

rtecall("filecopy") function

A RAD function that copies all of the data in a collection to another file variable. The database dictionary record for both the source and target files must exist. Records are only added.

Function

rtecall("filecopy")

Format

$L.success.flg=rtecall($L.fnc.name, $L.return.code, $L.dbdict.source.name, $L.dbdict.target.name, $L.count, $L.bad)

Parameters

The following parameters are valid for the rtecall("filecopy") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates if the function was successful.
$L.fnc.name	String	Name of the sub-function to call, in this case "filecopy".

$L.return.code	Number	Provides a more detailed return code.
$L.dbdict.source.name	String	The name of the source database dictionary record.
$L.dbdict.target.name	String	The name of the target database dictionary record.
$L.count	Number	A count of the number of records successfully moved.
$L.bad	Number	A count of the number of errors encountered.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.dbdict.source.name="location"
$L.dbdict.target.name="locationbak"
$L.success.flg=rtecall("filecopy", $L.return.code,
$L.dbdict.source.name,
$L.dbdict.target.name, $L.count, $L.bad)
```

Returns the "locationbak" file as an exact copy of the "location" file.

rtecall("fileinit") function

A RAD function that initializes a file variable in $targetfile from $sourcefile.

Function

rtecall("fileinit")

Format

rtecall($L.fnc.name, $errcode, $targetfile, $sourcefile)

Parameters

The following parameters are valid for the rtecall("fileinit") function:

Parameter	Data type	Description
$L.fnc.name	String	Name of the sub-function to call, in this case "fileinit".
$L.errcode	Number	The number of the error code encountered.
$L.targetfile	String	The target file in which to initialize the variable.
$L.sourcefile	String	The source file that contains the variable.

Factors

- The file initialized is identified by the *$sourcefile* variable.
- The current record for $targetfile is the same as the current record in $sourcefile.
- Update and delete operations cannot be performed against $targetfile.

Example

```
$L.void=rtecall("fileinit", $L.errcode, $file.old, $file)
```

rtecall("filldate") function

A RAD function that places the current date and time in a field in the current record.

Function

rtecall("filldate")

Format

$L.success.flg= rtecall($L.fnc.name, $L.return.code, $L.file, $L.field.name)

Parameters

The following parameters are valid for the rtecall("filldate") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates if the function was successful.
$L.fnc.name	String	The name of the sub-function to call, in this case "filldate".
$L.return.code	Number	Provides a more detailed return code.
$L.file	File	The file handle.
$L.field	String	The name of the field to be updated.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.file={[, , {}, , , {}, , , {}, , , {}, , , {}, {}, {}, {{[, , ,
]}}, , , ]} where the first field is
named "date"
$L.field.name="date"
$L.success.flg= rtecall("filldate", $L.return.code, $L.file,
$L.field.name)
$L.file returns ={['04/14/2000 08:27:19', , {}, , , {}, , , {}, , ,
{}, , , {}, {}, {}, {{[, , , ]}}, , ,
]}
```

rtecall("getnumber") function

A RAD function that replaces the getnumb RAD application.

Function

rtecall("getnumber")

Format

$L.flg=rtecall($L.fnc.name, $L.return.code, $L.number, $L.class, $L.field)

Parameters

The following parameters are valid for the rtecall("getnumber") function:

Parameter	Data type	Description
$L.flg	Logical	The value is *true* for success and *false* if an error occurs.
$L.fnc.name	String	The name of the sub-function to call, in this case "getnumber".
$L.return.code	Number	The value is zero (*0*) if everything is correct, negative one (*-1*) if an error occurs. This is related directly to $L.flg.
$L.number	String	The number or string returned
$L.class	String	The class of number for which you are searching (from the number file).
$L.field	String	Reserved for future use.

Factors

A sequential overview:

1. The function establishes a lock called "getnumb"+$L.class+$L.field and waits until the lock is established.
2. If the number class is not found, or if there are duplicates, the following error displays: " ($L.flg=false)"
3. The function reads the current number.
4. The function increments or decrements by the step value. The step defaults to one (*1*).
5. If incrementing and the number is greater than the reset value, it uses the start number. Start defaults to zero (*0*).
6. A new field called string.flg.
   - If string.flg is *false*, it remains a numeric type.
   - If string.flg is *true*, the number is converted to a string.
   - If string.flg is *unknown*, or the field isn't in the database dictionary (not added during an upgrade), the function checks for a prefix, a suffix, or length. If any one of these exists, the function converts the value to a string. If none of these exist the function leaves the value as a number.
7. To convert the value to a string, the function uses the prefix, corrects the length of the number (padded with zeros on the left), and then adds the suffix.
8. The function eventually saves the new number to the file and returns it to the user, converted to a string if necessary.
9. Finally, it unlocks and returns.

rtecall("getrecord") function

A RAD function that retrieves the record identified by Unique key values in $L.array.

Function
rtecall("getrecord")

Format
$L.void=rtecall($L.fnc.name, $L.errcode, $L.array, $L.file)

Parameters
The following parameters are valid for the rtecall("getrecord") function:

Parameter	Data type	Description
$L.fnc.name	String	The name of the sub-function to call, in this case "getrecord".
$L.errcode	Number	The number of the error code encountered.
$L.array	String	The name of the record to retrieve.
$L.file	String	The name of the table to retrieve the record from.

Factors
$L.array is an *input* variable, typically the key values returned by $L.array in the rtecall ("getunique") function.

Example

```
$L.void=rtecall("getrecord", $L.errcode, {"IM1001"}, $L.file)
```

Where:
The value of *$L.file* is initialized for the probsummary table.

rtecall("getunique") function

A RAD function that returns an array containing the Unique key values for a current record. The keys can later be used to retrieve the record using the rtecall("getrecord") function.

Function
rtecall("getunique")

Format
$L.void=rtecall($L.fnc.name, $L.errcode, $L.array, $L.file)

Parameters
The following parameters are valid for the rtecall("getunique") function:

Variable/Value	Description
$L.worked = true	If key values are returned in $L.keyvalues. For example, if $L.file represents a current database record.
$L.worked = false	If no key values are returned. For example, if $L.file does not represent a current database record.
$L.errcode = 0	The function succeeded.
$L.errcode =	The $L.file does not represent a file variable.

1	
$L.errcode = 2	The file identified by $L.file does not have a Unique key defined.
$L.file	The file variable initialized by the **rinit** command panel and containing a current record.

Factors

$L.keyvalues is an *output* variable. The output returns an array containing the unique key values for the record in $L.file. For example, if $L.file is for the problem table, an example of the key values returned in $L.keyvalues is *{"IM1001",1}*.

Example

```
$L.worked = rtecall( "getunique", $L.errcode, $L.keyvalues, $L.file
)
```

rtecall("isalnum") function

A RAD function that verifies that a string contains only alphabetic, numeric, or the non-alphabetic characters provided.

Function

rtecall("isalnum")

Format

$L.success.flag= rtecall ($L.fnc.name, $L.return.code, $L.str, $L.chars)

Parameters

The following parameters are valid for the rtecall("isalnum") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates if the function was successful.
$L.fnc.name	String	Name of the sub-function to call, in this case "isalnum".
$L.return.code	Number	Provides a more detailed return code.
$L.str	String	The string to be checked.
$L.chars	String	An optional comma-delimited string of non-numeric characters to allow.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.success.flag=true
$L.return.code=0
$L.str=phone in $L.file
$L.chars=" , (, ), -"
```

```
 $L.success.flag=rtecall("isalnum", $L.return.code, $L.str,
$L.chars)
```

Results:

- When phone in $L.file="(508) 362.2701" $L.success.flag is *false*.
- When phone in $L.file="(508) 362-2701" $L.success.flag is *true*.

rtecall("isalpha") function

A RAD function that verifies that a string contains only alphabetic characters or only alphabetic characters and the non-alphabetic characters provided.

Function

rtecall("isalpha")

Format

$L.success.flag= rtecall ($L.fnc.name, $L.return.code, $L.str, $L.chars)

Parameters

The following parameters are valid for the rtecall("isalpha") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates whether or not the function was successful.
$L.fnc.name	String	The name of the sub-function to call, in this case "isalpha".
$L.return.code	Number	Provides a more detailed return code.
$L.str	String	The string to be checked.
$L.chars	String	An optional comma-delimited string of non-alphabetic characters to allow.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
 $L.success.flag=true
 $L.return.code=0
 $L.str=name in $L.file
 $L.chars=" "
 $L.success.flag=rtecall("isalpha", $L.return.code, $L.str,
$L.chars)
```

Results:

- When name in $L.file="my name" $L.success.flag is *true*.
- When name in $L.file="my first name" $L.success.flag is *true*.
- When name in $L.file="my 1st name" $L.success.flag is *false*.

rtecall("islower") function

A RAD function that verifies that a string contains only lowercase characters, or only lowercase characters and the provided non-lowercase characters.

Function

rtecall("islower")

Format

$L.success.flag= rtecall ($L.fnc.name, $L.return.code, $L.str, $L.chars)

Parameters

The following parameters are valid for the rtecall("islower") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates if the function was successful.
$L.fnc.name	String	Name of the sub-function to call, in this case "islower".
$L.return.code	Number	Provides a more detailed return code.
$L.str	String	The string to be checked.
$L.chars	String	An optional comma-delimited string of non-lowercase characters to allow.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
 $L.success.flag=true
 $L.return.code=0
 $L.str=name in $L.file
 $L.chars=" "
 $L.success.flag=rtecall("islower", $L.return.code, $L.str,
$L.chars)
```

Results:

- When name in $L.file="b1b" $L.success.flag is *false*.
- When name in $L.file="Bob" $L.success.flag is *false*.
- When name in $L.file="bob" $L.success.flag is *true*.

rtecall("isnumeric") function

A RAD function that verifies that a string contains only numeric characters, or only numeric characters and the provided non-numeric characters.

Function

rtecall("isnumeric")

Format

$L.success.flag= rtecall ($L.fnc.name, $L.return.code, $L.str, $L.chars)

Parameters

The following parameters are valid for the rtecall("isnumeric") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates if the function was successful.
$L.fnc.name	String	Name of the sub-function to call, in this case "isnumeric".
$L.return.code	Number	Provides a more detailed return code.
$L.str	String	The string to be checked.
$L.chars	String	An optional comma-delimited string of non-numeric characters to allow.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
 $L.success.flag=true
 $L.return.code=0
 $L.str=name in $L.file
 $L.chars="-,+,*,/,%,$"
 $L.success.flag=rtecall("isnumeric", $L.return.code, $L.str,
$L.chars)
```

Results:

- When name in $L.file="B1B" $L.success.flag is *false*.
- When name in $L.file="Bob" $L.success.flag is *false*.
- When name in $L.file="36" $L.success.flag is *true*.

rtecall("isupper") function

A RAD function that verifies that a string contains only uppercase characters, or only uppercase characters and the provided non-uppercase characters.

Function

rtecall("isupper")

Format

$L.success.flag= rtecall ($L.fnc.name, $L.return.code, $L.str, $L.chars)

Parameters

The following parameters are valid for the rtecall("isupper") function:

Parameter	Data type	Description

$L.success.flg	Logical	Indicates if the function was successful.
$L.fnc.name	String	Name of the sub-function to call, in this case "isupper".
$L.return.code	Number	Provides a more detailed return code.
$L.str	String	The string to be checked.
$L.chars	String	An optional comma-delimited string of non-upper case characters to allow.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
 $L.success.flag=true
 $L.return.code=0
 $L.str=name in $L.file
 $L.chars=" "
 $L.success.flag=rtecall("isupper", $L.return.code, $L.str,
$L.chars)
```

Results:

- When name in $L.file="B1B" $L.success.flag is *false*.
- When name in $L.file="Bob" $L.success.flag is *false*.
- When name in $L.file="BOB" $L.success.flag is *true*.

rtecall("log") function

A RAD function that sends a message to the external sm.log file.

Function

rtecall("log")

Format

$L.success.flg=rtecall($L.fnc.name, $L.return.code, $L.message)

Parameters

The following parameters are valid for the rtecall("log") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates whether or not the function was successful.
$L.fnc.name	String	The name of the sub-function to call, in this case "log".
$L.return.code	Number	Provides a more detailed return code.
$L.message	String	The message to be sent.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.success.flg=rtecall("log", $L.return.code, "This is a new
message")
```

Generates the following **bold** line in the sm.log:

```
   137 04/12/2000 07:03:44 System background scheduler: event
started at:
04/11/2000 23:02:48.
   137 04/12/2000 07:03:45 System background scheduler: availability
started at:
04/11/2000 23:02:49.
   137 04/12/2000 07:03:45 System background scheduler: contract
started at:
04/11/2000 23:02:50.
   137 04/12/2000 07:03:45 System background scheduler: ocm started
at:
04/11/2000 23:02:51.
   137 04/12/2000 07:03:45 System startup completed successfully,
elapsed time:
00:00:18.
   137 04/12/2000 07:32:48 24 records from location unloaded to:
locations.save,
00:00:01 elapsed.
   306 04/12/2000 09:05:09 This is a new message
```

rtecall("passchange") function

A RAD function that changes a user password.

Function

rtecall("passchange")

Format

$L.success.flag=rtecall($L.fnc.name, $L.return.code, $L.old.pass, $L.new.pass, $L.confirm.pass)

Parameters

The following parameters are valid for the rtecall("passchange") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates if the function was successful.
$L.fnc.name	String	Name of the sub-function to call. In this case, the sub-function called is passchange.
$L.return.code	Number	Provides a more detailed return code.
$L.old.pass	String	The old password.
$L.new.pass	String	The new password.

$L.confirm.pass	String	A confirmation of the new password.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.success.flg= rtecall("passchange", $L.return.code, "oldpassword",
"newpassword", "newpassword")
```

rtecall("policycheck") function

A RAD function that imposes data policy as defined in the System Definition.

Function

rtecall("policycheck")

Format

$L.success.flag= rtecall($L.fnc.name, $L.return.code, $L.file)

Parameters

The following parameters are valid for the rtecall("policycheck") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates if the function was successful.
$L.fnc.name	String	Name of the sub-function to call, in this case "policycheck".
$L.return.code	Number	Provides a more detailed return code.
$L.file	String	The file handle.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.rc=rtecall("policycheck", $L.errcode, $L.filed)
```

rtecall("qbeform") function

A RAD function that returns a record list form, which can be passed into a rio (Record Display/Input) Command panel or fdisp panel. You can insert this function into a format file handle using the contents() function. The following example shows the syntax to use in the contents function: `contents($L.format) = $L.qbe.form`

Function

rtecall("qbeform")

Format

$L.success.flg= rtecall($L.fnc.name,$L.return.code,$L.file,$L.qbe.form)

Parameters

The following parameters are valid for the rtecall("qbeform") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates whether or not the function was successful.
$L.fnc.name	String	The name of the subfunction to call; in this case "qbeform".
$L.return.code	Number	Provides a more detailed return code.
$L.file	File	The file handle used to generate the record list. For example, obtain the contacts table by using the "rinit" panel.
$L.qbe.form	Structure	The resulting record list form returned as a structure.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.success.flg= rtecall("qbeform",$L.return.code,$L.file,
$L.qbe.form)
```

rtecall("recdupl") function

A RAD function that copies the contents of the current record into the contents of another record.

Function

rtecall("recdupl")

Format

rtecall($L.fnc.name, $return.code, $targetfile, $sourcefile)

Parameters

The following parameters are valid for the rtecall("recdupl") function:

Parameter	Data type	Description
$L.fnc.name	String	Name of the sub-function to call, in this case "recdupl".
$L.return.code	Number	Provides a more detailed return code.
$targetfile	String	The target record to which fields are copied.
$sourcefile	String	The source file from which fields are copied.

Factors

- Both *$targetfile* and *$sourcefile* must have identical descriptors. Data is copied by position, not field name.
- Both *$targetfile* and *$sourcefile* must be initialized file variables.
- The update and delete operations cannot be performed against *$targetfile*.

Example

```
$L.flg=rtecall("recdupl", $L.return.code, $L.temp, $L.file)
```

rtecall("recordexists") function

Specifies whether any records exist that match a particular query regardless of any Mandanten data restrictions. This call does not return any records from the query.

Function

rtecall("recordexists")

Format

$Lexists=rtecall("recordexists", $L.rc, $L.filename, $L.query)

Parameters

The following parameters are valid for the rtecall("recordexists") function:

Parameter	Data type	Description
"recordexists"	String	Determines the name of the RTE call, in this case "recordexists".
$L.rc	String	Provides a more detailed return code. This parameter is not currently used but is reserved for future use.
$L.filename	String	The name of the Service Manager table to query. For example, contacts or probsummary.
$L.query	String	The query to be run. The query can be true, false, a string, or a parsed expression. This query ignores all Mandanten restrictions.

Factors

The call returns true if at least one record matches the query.

The call returns false if there are no records that match the query.

Example

```
rtecall( "recordexists", $L.rc, "contacts", true )
```

Returns true if there are any records in the contacts table.

```
rtecall( "recordexists", $L.rc, "device", "logical.name=\"pc001\"" )
```

Returns true if there is a record in the device table with a logical name of "pc001" regardless of whether the current user has access to the record or not.

rtecall("refresh") function

This function retrieves the most current value of a record from the database and returns it in the file variable.

Function

rtecall("refresh")

Format

$L.success.flg=rtecall("refresh", $L.errcode, $L.file)

Parameters

The following parameters are valid for the rtecall("refresh") function:

Variable/Value	Description
$L.errcode = 0	The function succeeded.
$L.errcode = 1	The $L.file does not represent a file variable.
$L.file	The file variable initialized by the **rinit** command panel. It contains the current record.

Factors

If the $L.success.flg is false, the function failed. It is true, the function succeeded..

Example

```
$L.success.flg = rtecall( "refresh", $L.errcode, $L.file )
```

rtecall("rfirst") function

A RAD function that places the pointer at the first record in a record list.

Function

rtecall("rfirst")

Format

$L.success.flg=rtecall($L.fnc.name, $L.return.code, $L.file)

Parameters

The following parameters are valid for the rtecall("rfirst") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates if the function was successful.
$L.fnc.name	String	The name of the sub-function to call, in this case "rfirst".
$L.return.code	Number	Provides a more detailed return code.
$L.file	File	A file handle that represents the collection.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.success.flg=rtecall("rfirst",$L.return.code, $L.file)
```

rtecall("rgoto") function

A RAD function that places the pointer at the indicated record.id in a record list.

Function
rtecall("rgoto")

Format
$L.success.flg=rtecall($L.fnc.name, $L.return.code, $L.file, $L.record.id)

Parameters
The following parameters are valid for the rtecall("rgoto") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates whether or not the function was successful.
$L.fnc.name	String	The name of the sub-function to call, in this case "rgoto".
$L.return.code	Number	Provides a more detailed return code.
$L.file	File	A file handle that represents the collection.
$L.record.id	Number	A record number.

Factors
If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.success.flg=rtecall("rgoto",$L.return.code, $L.file,
$L.record.id)
```

rtecall("rid") function

A RAD function that returns the record number of the current record, as represented by $L.file.

Function
rtecall("rid")

Format
$L.success.flg=rtecall($L.fnc.name, $L.return.code, $L.file, $L.record.id)

Parameters
The following parameters are valid for the rtecall("rid") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates whether or not the function was successful.
$L.fnc.name	String	The name of the sub-function to call, in this case "rid".
$L.return.code	Number	Provides a more detailed return code.
$L.file	File	A file handle that represents the collection.
$L.record.id	Number	A record number.

Factors
If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.success.flg=rtecall("rid",$L.return.code, $L.file, $L.record.id)
```

rtecall("scantable") function

A RAD function that scans the specified table for certain types of invalid records and fixes the records.

Function

rtecall("scantable")

Format

$L.success.flg=rtecall($L.fnc.name, $return.code, $scantype, $tablename, $bfix, $columns)
$L.success.flg=rtecall($L.fnc.name, $return.code, $scantype, $tablename, $bfix, $totalnum, $fixed, $deleted)

Parameters

The following parameters are valid for the rtecall("scantable") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates if the function was successful.
$L.fnc.name	String	Name of the sub-function to call, in this case "scantable".
$L.return.code	Number	Provides a more detailed return code.
$scantype	Number	Specifies the type of scan the function will perform, which can be one of the following: • 1–to scan for duplicate records on specified columns • 2–to scan for null values on non-nullable fields, such as fields with a "Unique" or "No Nulls" key applied • 3–to scan for values whose data types do not match the data type defined on the field
$L.tablename	String	The name of the table to be scanned.
$bfix	Bool	Specifies whether to automatically fix the invalid data.
$columns	String array	Only available when $scantype is 1, specifies the columns on which the function will scan for duplicate records.
$totalnum	Number	Indicates the number of the invalid records that were found.
$fixed	Number	Indicates the number of the invalid records that were fixed.

$deleted	Number	Indicates the number of the records that were removed.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded. $totalnum, $fixed, and $deleted are available when $scantype is 2 or 3. This total number should include $fixed, $deleted, and the number of invalid records that failed to be fixed.

Examples

The following function call scans the **ioaction** table for duplicate records on the **field1** and **field2** fields and fixes the duplicate records.

```
$L.flg=rtecall("scantable", $L.return.code, 1, "ioaction", true,
{"field1","field2"})
```

The following function call scans the **ioaction** table for null values that are disallowed by keys.

```
$L.flg=rtecall("scantable", $L.return.code, 2, "ioaction", true,
$all, $fixed, $deleted)
```

Additional information

If you choose to fix invalid data by setting **$bfix** to 1:

- In a "type-1" scan, the function differentiates duplicate items by appending suffixes such as -dup1, -dup2...-dup*x*.
- In a "type-2" scan, the function:
  - removes the record if null values violate a "Unique" key, or
  - replaces the first of the null values with a default value if null values violate a "No Nulls" key.
- In a "type-3" scan, the function:
  - converts the value into the correct data type if the value data makes sense in the target data type
  - replaces the value with a default value in the correct data type.

rtecall("sort") function

A RAD function that sorts a list or a list of lists in ascending or descending order.

Function

rtecall("sort")

Format

$L.success.flg=rtecall($L.fnc.name, $L.return.code, $L.grid, $L.index, $L.type, $L.nulls)

Parameters

The following parameters are valid for the rtecall("sort") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates whether or not the function was successful.

$L.fnc.name	String	Name of the sub-function to call, in this case "sort".
$L.return.code	Number	Provides a more detailed return code.
$L.grid	Array	The list or list of lists to be sorted.
$L.index	Number	The index of the array in the grid to use as the sort field. A zero (*0*) is the first element of the array.
$L.type	Number	The type of sort to perform. Use zero (*0*) for an ascending sort or one (*1*) for a descending sort.
$L.nulls	Number	Specifies how the function should sort null values. Use one (*1*) to sort null values at the end of the list or zero (*0*) to sort values at the start of the list.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Examples

```
$L.list={29, 2, 72, 42}
$L.success.flg=rtecall("sort", $L.return.code, $L.list, 1, 0)
Returns: $L.list={2, 29, 42, 72}
```

```
$L.list={29, 2, 72, 42}
$L.success.flg=rtecall("sort", $L.return.code, $L.list, 1, 1)
Returns: $L.list={72, 42, 29, 2}
```

```
$L.list={29, 2, , 72, 42}
$L.success.flg=rtecall("sort", $L.return.code, $L.list, 1, 0, 1)
Returns: $L.list={2, 29, 42, 72, }
```

```
$L.list={29, 2, , 72, 42}
$L.success.flg=rtecall("sort", $L.return.code, $L.list, 1, 0, 0)
Returns: $L.list={, 2, 29, 42, 72}
```

```
$L.list={{"a", "b", "d", "c"},{1, 3, 4, 2}}
$L.success.flg=rtecall("sort", $L.return.code, $L.list, 1, 0)
Returns: $L.list= {{"a", "c", "b", "d"}, {1, 2, 3, 4}}}
```

```
$L.list={{"a", "b", "d", "c"},{1, 3, 4, 2}}
$L.success.flg=rtecall("sort", $L.return.code, $L.list, 1, 1)
Returns: $L.list={{"d", "b", "c", "a"}, {4, 3, 2, 1}}}
```

```
$L.list={{"a", "b", "d", "c"},{1, 3, 4, 2}}
$L.success.flg=rtecall("sort", $L.return.code, $L.list, 0, 0)
Returns: $L.list={{"a", "b", "c", "d"}, {1, 3, 2, 4}}}
```

```
$L.list={{"a", "b", "d", "c"},{1, 3, 4, 2}}
$L.success.flg=rtecall("sort", $L.return.code, $L.list, 0, 1)
Returns: $L.list={{"d", "c", "b", "a"}, {4, 2, 3, 1}}
```

rtecall("statusupdate") function

A RAD function that dynamically changes the caption of a widget on the current format. The widget must have a name property specified. This function is commonly used to update the format during a long-running transaction to inform the user of the status of the current operation. For example, converting a file to an RDBMS.

Function

rtecall("statusupdate")

Format

$L.success.flag = rtecall($L.func.name, $L.return.code, $L.widget.name, $L.widget.caption

Parameters

The following parameters are valid for the rtecall("statusupdate") function:

Parameter	Data type	Description
$L.widget.name	String	The name of the form widget, as specified in the name property of the widget.
$L.widget.caption	String	A string to be displayed in the widget.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
L.success.flg - rtecall("statusupdate", $L.return,
"MyScrollingMarqueeWidget", "The end user
       will see this text in the marquee" )
```

rtecall("transtart") function

A RAD function that measures the amount of data transferred, elapsed time, and CPU usage of any transaction. This function is commonly invoked from the RAD debugger and used in conjunction with the rtecall ("transtop") function.

Function

rtecall("transtart")

Format

$L.success.flag= rtecall($L.fnc.name, $L.return.code, $L.transaction)

Parameters

The following parameters are valid for the rtecall("transtart") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates whether or not the function was successful.
$L.fnc.name	String	Name of the sub-function to call, in this case "transtart".

$L.return.code	Number	Provides a more detailed return code.
$L.transaction	String	Any name selected by the user for the transaction to measure.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.success.flg= rtecall("transtart", $L.return, "problemopen")
```

rtecall("transtop") function

A RAD function that measures the amount of data transferred, elapsed time, and CPU usage of any transaction. The function is commonly invoked from the RAD debugger and used in conjunction with the rtecall("transtart") function.

- The CPU time is retrieved from the server machine.
- You can have multiple transaction timings active at the same time.

Function

rtecall("transtop")

Format

$L.success.flag= rtecall($L.fnc.name, $L.return.code, $L.transaction, $results)

Parameters

The following parameters are valid for the rtecall("transtop") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates whether or not the function was successful.
$L.fnc.name	String	Name of the sub-function to call, in this case "transtop".
$L.return.code	Number	Provides a more detailed return code.
$L.transaction	String	Any name selected by the user for the transaction to measure.
$L.$results	Array	The results gathered, in the form of an array: {elapsed seconds, cpu seconds, 0, 0, 0}. Service Manager no longer keeps track of the third, fourth, and fifth value in the array and it always reports as zero.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.success.flg=rtecall("transtop", $L.return.code, "problemopen",
$results)
$results={58.164, 0.961, 0, 26712, 2477}
```

This example shows that the "problemopen" transaction took 58.164 seconds of real time and

0.961 CPU seconds.

rtecall("trigger") function

A RAD function that turns triggers on or off for the current session. Other Service Manager users are not affected.

Function

rtecall("trigger")

Format

$L.success.flag= rtecall($L.fnc.name, $L.return.code, $L.switch)

Parameters

The following parameters are valid for the rtecall("trigger") function:

Parameter	Data type	Description
$L.success.flg	Logical	Indicates if the function was successful.
$L.fnc.name	String	Name of the sub-function to call, in this case "trigger".
$L.return.code	Number	Provides a more detailed return code.
$L.switch	Number	The value is either one (*1*) to turn triggers on. or zero (*0*) to turn triggers off.

Factors

If the $L.success.flg is *false*, the function failed. If it is *true*, the function succeeded.

Example

```
$L.success.flg=rtecall("trigger", $L.return.code, 1) to turn
triggers on
$L.success.flg=rtecall("trigger", $L.return.code, 0) to turn
triggers off
```

List: RAD routines

This section provides a list of RAD routines, a brief definition of each, and a link to a detailed description of their use.

Routine	Description
as.copy	Copies an element from one position to another, leaving the original in place.
as.delete	Deletes an element from an array.
as.get.name	Returns the structure name of the arrayed structure in which the cursor is located.

as.insert	Inserts a NULL element.
as.move	Moves an element from one position in an array to another by creating a copy and then deleting the original
as.options	Displays all available arrayed structure options on the rio (Record Display/Input) Command panel.
as.sort	Sorts an arrayed structure based on the contents of a particular field within the arrayed structure.
axces.apm	Creates or updates an incident based on an Event Services eventin record.
axces.cm3	Creates or updates a change request based on an Event Services eventin record.
axces.cm3.cts.write	Creates a schedule record that processes alerts for Change Management records.
axces.database	Adds, updates or deletes records from Service Manager files using Event Services.
axces.email	Creates an eventout record based on the Service Manager mail record that was passed in.
axces.email.receive	Receives external e-mail and converts it to Service Manager mail using Event Services.
axces.fax	Builds an eventout record that can be processed by the SCAutomate / Fax application.
axces.move.intoout	Copies an eventin record to the eventout file and changes the evtype.
axces.page	Creates an eventout record based on information passed in that can be processed by SCAutomate / Pager to send a numeric or alphanumeric page.
axces.rm	Creates or updates Request Management quotes via Event Services eventin record.
axces.sap.hybrid.evin	Breaks events into appropriate constituent parts.
axces.software	Adds, updates, or deletes a software configuration item that an external agent (other than ServerView or StationView) discovers to the pcfiles file if the filter criteria are satisfied.
axces.sm	Creates or updates Service Desk interactions based on an eventin record using Event Services.
axces.write	Build the eventout record used by the SCAutomate interface.
database	Provides access to other tables from a particular record.
es.approval	Processes approvals for Request Management and Change

	Management.
fill.fc	Copies data from a target file to the current file, using predefined links under user control.
getnumb.fc	Establishes the numbering sequence in sequential numbering.
marquee.publish	Sends a marquee message.
marquee.send	Reads the schedule record and initiate the marquee.publish routine.
message.fc	Sends messages under user control.
post.fc	Executes the posting routine from the subroutines process (automatic posting) and from the additional options process (manual posting).
publish	Publishes a variable into SYSPUB.
query.stored	Accesses all queries for a specific file or to execute a specific stored query.
scauto.inventory	Provides access to configuration items.
sort.array	Sorts simple arrays (number, character, date/time) in either ascending or descending order.
validate.fields	Validates fields in a form.
wmi.inventory.checks	Checks the eventin record for status information from a type of scan. It also processes the information from the scan into the Configuration Management application using the scauto.inventory RAD routine.

RAD routine: as.copy

A RAD routine that copies an element within an array or structure from one position to another, leaving the original in place. This routine is used in conjunction with as.delete, as.insert, and as.move routines.

Routine

as.copy

Parameters

The following parameters are valid for the as.copy routine:

Name	Data type	Required	Description
file	record	No	The file variable that you are currently editing. The value is always *$file* when called from Format Control.
name	character	No	Data is not normally passed to this parameter. The default is the result of executing the function cursor.field.name(1). The

			application as.get.name then executes against this result. **Note**: You can pass the name of a specific arrayed structure. If the data passed to this parameter does not contain commas, it is considered to be an arrayed structure name. If a comma is found, the application as.get.name is executed against the data.
index	number	No	The number of lines (height property) in the subform that displays the arrayed structure. For example, if the arrayed structure subform contains data on lines 1 and 2, then this parameter is 2. The default value is 1. **Notes**: This is not the window size that displays the arrayed structure. When calling the application from Format Control, the value must be passed as a number by using the following syntax: `val("n", 1)` Where: The value of *n* = the number of lines in the subform, and *1* indicates a data type of number.

RAD routine: as.delete

A RAD routine that deletes an element from an array. This routine is used in conjunction with as.copy, as.insert, and as.move.

Routine

as.delete

Parameters

The following parameters are valid for the as.delete routine:

Name	Data type	Required	Description
file	record	No	The file variable currently being edited. The value is always *$file* when called from Format Control.
name	character	No	Data is not normally passed to this parameter. The default is the result of executing the function cursor.field.name(1). The application as.get.name then executes against this result. **Note**: You can pass the name of a specific arrayed structure. If the data passed to this parameter does not contain commas, it is considered to be an arrayed structure name. If a comma is found, the application as.get.name

			executes against the data.
index	number	No	The number of lines (height property) in the subform that displays the arrayed structure. For example, if the arrayed structure subform contains data on lines 1 and 2, then this parameter is 2. The default value is 1. **Notes**: This is not the window size that displays the arrayed structure. When calling the application from Format Control, the value must be passed as a number by using the following syntax: `val("n", 1)` Where: The value of *n* = the number of lines in the subform, and *1* indicates a data type of number.

RAD routine: as.get.name

A RAD routine that returns the structure name of the arrayed structure where the cursor is located.

Routine
as.get.name

Parameters
The following parameters are valid for the as.get.name routine:

Name	Data type	Required	Description
name	character	No	The structure name of the arrayed structure. The default is the result of executing the function cursor.field.name(1).
prompt	character	No	The returned structure name of the arrayed structure. This is a data return parameter, data should not be passed to it.

RAD routine: as.insert

A RAD routine used to insert a NULL element. It is used in conjunction with as.copy, as.delete, and as.move.

Routine
as.insert

Parameters

The following parameters are valid for the as.insert routine:

Name	Data type	Required	Description
file	record	No	The file variable currently being edited. The value is always *$file* when called from Format Control.
name	character	No	Data is not normally passed to this parameter. The default is the result of executing the function cursor.field.name(1). The application as.get.name then executes against this result. **Note**: You can pass the name of a specific arrayed structure. If the data passed to this parameter does not contain commas, it is considered an arrayed structure name. If a comma is found, the application as.get.name executes against the data.
index	number	No	The number of lines (height property) in the subform that displays the arrayed structure. For example, if the arrayed structure subform has data on lines 1 and 2, then this parameter is *2*. The default value is *1*. **Notes**: This is not the window size that displays the arrayed structure. When calling the application from Format Control, the value must be passed as a number by using the following syntax: `val("n", 1)` Where: The value of *n* = the number of lines in the subform, and *1* indicates a data type of number.

RAD routine: as.move

A RAD routine used to move an element from one position in an array to another by creating a copy and then deleting the original. This routine is used in conjunction with as.copy, as.insert, and as.delete.

Routine

as.move

Parameters

The following parameters are valid for the as.move routine:

Name	Data type	Required	Description
file	**record**	No	The file variable currently being edited. The value is always

			$file when called from Format Control.
name	character	No	Data is not normally passed to this parameter. The default is the result of executing the function cursor.field.name(1). The application as.get.name then executes against this result. **Note**: You can pass the name of a specific arrayed structure. If the data passed to this parameter does not contain commas, it is considered an arrayed structure name. If a comma is found, the application as.get.name executes against the data.
index	number	No	The number of lines (height property) in the subform that displays the arrayed structure. For example, if he arrayed structure subform has data on lines 1 and 2, then this parameter is *2*. The default value is *1*. **Notes**: This is not the window size that displays the arrayed structure. When calling the application from Format Control, the value must be passed as a number by using the following syntax: `val("n", 1)` Where the value of *n* = the number of lines in the subform, and *1* indicates a data type of number.

RAD routine: as.options

A RAD routine used to display all available arrayed structure options one *rio* (Record Display/Input) Command panel.

Note: The as.options routine provides the control needed to copy, insert, delete, and move fields. It is not necessary to run as.copy, as.move, as.insert, and as.delete in conjunction with this routine.

Routine

as.options

Parameters

The following parameters are valid for the as.options routine:

Name	Data type	Required	Description
file	record	No	The file variable currently being edited. The value is always *$file* when called from Format Control.
name	character	No	Passes the name of the current form. If left blank, the default is the result of executing the function current.format().

cond.input	Boolean	No	Allows the user to edit the data record when the Arrayed structure maintenance options are displayed. The default value is *true*. **Note**: When calling the application from Format Control, the value must be passed as a data type of 4 (logical) by using the following syntax: `val("false", 4)`
index	number	No	The number of lines in the subform that displays the arrayed structure. For example, if the arrayed structure subform has data on lines 1 and 2, then the parameter is 2. The default value is *1*. **Notes**: This is not the window size that displays the arrayed structure. When calling the application from Format Control, the value must be passed as a number by using the following syntax: `val("n", 1)` Where: The value of *n* = the number of lines in the subform, and one (*1*) indicates a data type of number.

RAD routine: as.sort

A RAD routine used to sort a structure that includes an array. It sorts based on the contents of a particular field within the array of the structure. This routine supports major, minor, and intermediate sorting by allowing you to sort multiple array fields in the structure with one pass of the routine. You can also specify an ascending or descending sort order. You must call this routine from Format Control or RAD on an as-needed basis.

Routine

as.sort

Parameters

The following parameters are valid for the as.sort routine:

Name	Data type	Required	Description
file	record	No	The file variable currently being edited. The value is always *$file* when called from Format Control.
name	character	No	The name of the structure you want to sort. This name

			corresponds to the input field of the subform object in the form to which the Format Control record is attached. **Note**: If the data passed to this parameter does not contain commas, it is considered an arrayed structure name. If a comma is found, the application as.get.name executes against the data.
numbers	array (numbers)	Yes	Specifies the field numbers on which to sort within the arrayed structure. In Format Control, you must first define a *$variable* in the Initializations form for the array of numbers. Next, pass that variable to the numbers input field in the Additional Options definition. **Note**: Field numbers correspond to the index numbers listed in the database dictionary and not to the sequence in which they appear on the form.
condition	array (Boolean)	No	Specifies either ascending or descending order for each field being sorted in the arrayed structure, as specified in the numbers parameter. To specify an ascending order, pass a value of *true*. To specify descending order, pass a value of *false*. The default value is *true*. Each element in this array has a direct correlation with the elements of the numbers parameter. In Format Control, you must first define a *$variable* in the Initializations screen that defines the Boolean array. Next, pass the variable to this input field in the Additional Options definition.

RAD routine: axces.apm

A RAD routine that creates or updates an incident based on an Event Services eventin record.

Routine
axces.apm

Parameters
The following parameters are valid for the axces.apm routine:

Name	Data type	Required	Description
record	record	Yes	Identifies the **eventin** record to pass.
text	character	No	Provides the specific action to take.

prompt	character	No	Indicates the Map Name used in event registration.
query	character	No	Indicates the query to use to find the pre-existing record updated by this Input event.
string1	character	No	Identifies the Service Manager file where the data is recorded.
boolean1	Boolean	No	An error occurs if this is not set. If the value is set to *NULL*, change to *close*. If evstatus begins with error, the value is *false*. If evstatus does not begin with error, the value is *true*.

RAD routine: axces.cm3.cts.write

A RAD routine that creates a schedule record that processes alerts for Change Management records.

Routine

axces.cm3.cts.write

Parameters

The following parameters are valid for the axces.cm3.cts.write routine:

Name	Data type	Required	Description
record	record	No	Indicates what file variable passes to the application.
name	character	No	Indicates the System ID of the system to send the message to and where to calculate the schedule time.
prompt	character	No	Indicates the Client of the system to send the message to and where to calculate the schedule time.
time1	time	No	What is the target time for this event? Use the tod() function to indicate to the system that the message sends at the next acceptable time.
query	character	No	Once you find the next acceptable time for this System ID and client combination, a new schedule record generates for a message of this type.
boolean1	Boolean	No	When set to *true*, the rescheduling portion of the code is bypassed and all messages occur as soon as possible.

RAD routine: axces.cm3

A RAD routine that creates or updates a change request based on an Event Services eventin record.

Routine

axces.cm3

Parameters

The following parameters are valid for the axces.cm3 routine:

Name	Data type	Required	Description
string1	character	No	Identifies the area of CM3 to work in and where to record data for branching or changing tasks (cm3r or cm3t).
record	record	No	Indicates eventin record to process.
prompt	character	No	What map interprets the incoming message?
text	character	No	What map interprets the incoming message?
boolean1	Boolean	No	Are Output events created?
string1	character	No	Identifies the area of CM3 to work in and where to record data for branching or changing tasks (cm3r or cm3t).
query	character	No	The query for existing change record.

RAD routine: axces.database

A RAD routine called by many Event Services events to add, update or delete records from Service Manager tables using Event Services.

Routine

axces.database

Parameters

The following parameters are valid for the axces.database routine:

Parameter Name	Data type	Required	Description
record	file record	Yes	Identifies the eventin record to pass.
prompt	character	Yes	Identifies the Map name used in Event Registration. The event map name must end with **a** for add events and must end with **u** for update events.
string1	character	Yes	Identifies the Service Manager table to populate with data.
text	character	Yes	Provides the specific action to take, such as the following: • add • delete • update

query	character	No	Indicates the query to use to find the existing record to update.
boolean1	Boolean	No	When flagged, the value is *true* and Service Manager writes an eventout record. When not flagged, the value is *false* and no eventout record is written.
cond.input	Boolean	No	Specifies the behavior to follow when more than one record matches the query. When flagged *true*, Service Manager processes more than one record. When not flagged (value is *false*), Service Manager processes only the first record found.
name	character	No	Indicates the Format Control record to use.

RAD routine: axces.email

A RAD routine that creates an eventout record based on the Service Manager mail record that was passed in.

Routine

axces.email

Parameters

The following parameters are valid for the axces.email routine:

Name	Data type	Required	Description
record	record	No	Writes the unique record identifier to the eventin record. Set up the *$axces* variable using Format Control to create the intended result.
text	character	No	Indicates the delimiter character.

RAD routine: axces.email.receive

A RAD routine that receives external e-mail and converts it to Service Manager mail using Event Services.

Routine

axces.email.receive

Parameters

The following parameter is valid for the axces.email.receive routine:

Name	Data type	Required	Description
record	record	No	This parameter writes the unique record identifier to the eventin record. Set up the *$axces* variable using Format Control to create the intended result.

RAD routine: axces.fax

A RAD routine that builds an eventout record that can be processed by the SCAutomate / Fax application.

Routine
axces.fax

Parameters
The following parameters are valid for the axces.fax routine:

Name	Data type	Required	Description
names, 1	array (characters)	No	Passes the name of the sender. Use your login ID.
name	character	No	Passes the name of the recipient. Use the recipient's login ID.
prompt	character	No	Defines the recipient's fax telephone number. For example, fax.phone from the contacts table.
string1	character	No	Defines the separation character. If you define your own character, ensure that it does not occur naturally in fields in the event. The default is ^.
text	character	No	Names the form or text string. If a record variable passes in the record parameter, pass the form name in the text parameter. If you pass a string in the text parameter, use the pipe symbol (|) to separate line of text.
names, 2	array (characters)	No	Defines the FAX title.
names, 3	array (characters)	No	The destination of the FAX phone number.
record	record	No	Passes the record variable. Use $file in Format Control.

RAD routine: axces.move.intoout

A RAD routine that copies an eventin record to the eventout file and changes the evtype.

Routine
axces.move.intoout

Parameters

The following parameters are valid for the axces.move.intoout routine:

Name	Data type	Required	Description
record	record	Yes	Identifies the eventin record to pass.
name	character	No	Identifies the eventout type.

RAD routine: axces.page

A RAD routine that creates an eventout record based on information passed in that can be processed by SCAutomate / Pager to send a numeric or alphanumeric page.

Routine

axces.page

Parameters

The following parameters are valid for the axces.page routine:

Name	Data type	Required	Description
name	character	Yes	Contains the Contact Name from the contacts table.
prompt	character	Yes, if the text parameter is not used.	The numeric message.
text	character	Yes, if the prompt parameter is not used.	The alphanumeric message.
string1	character	No	Defines the separation character. If you define your own character, ensure that it does not occur naturally in fields in the event. The default value is the caret character (^).
query	character	No	Defines the page response code used by the pageresp input event to identify the type of event processing to occur. For example, to update a particular incident with the response from a page, pass *pm* and the incident number, such as pm9700123. The registration record then determines the application to call by examining the data in the first position of the evfields field.
values	array (characters)	No	Identifies a list of addressees from the contacts table or operator table.

names, 1	array (characters)	No	Lists the pager telephone number from the contacts table.
names, 2	array (characters)	No	Lists the pager PIN number from the contacts table.
names, 3	array (characters)	No	Passes the name of a group defined in the distgroup table.

RAD routine: axces.rm

A RAD routine that creates or updates Request Management quotes via Event Services eventin record.

Routine

axces.rm

Parameters

The following parameters are valid for the axces.rm routine:

Name	Data type	Required	Description
record	record	No	This parameter identifies which eventin record to pass.
text	character	No	Provides the specific action to take, such as add, open, update, close, approve, disapprove or unapprove.
prompt	character	No	Indicates the Map name in event registration.
query	character	No	The query string used to select a pre-existing record to update.
string1	character	No	Identifies the Service Manager table (ocmq, ocmo, or ocml) where data is recorded.
boolean1	Boolean	No	The logical parameter flag, *true* or *false*, that indicates whether to write the eventout record on transaction completion.
name	character	No	Identifies the operator (security profile) in Service Manager to use and which userID to stamp on records that are processed using this event.

RAD routine: axces.sap.hybrid.evin

A RAD routine that splits events out into the appropriate constituent parts.

Routine

axces.sap.hybrid.evin

Parameters

The following parameters are valid for the axces.sap.hybrid.evin routine:

Name	Data type	Required	Description
record	record	No	A hybrid event that calls the appropriate routines for header and detail events. This parameter expects one header and any number of properly-formed detail events.
prompt	character	No	Identifies the eventregister that interprets the detail portion of the event message.
name	character	No	Identifies the eventregister that interprets the header portion of the event stream.

RAD routine: axces.sm

A RAD routine that creates or updates Service Desk interactions based on an eventin record using Event Services.

Routine

axces.sm

Parameters

The following parameters are valid for the axces.sm routine:

Name	Data type	Required	Description
record	record	Yes	Identifies the eventin record to pass.
prompt	character	No	Indicates the Map name used in event registration.
string1	character	No	Identifies the Service Manager file where data is recorded.
query	character	No	Indicates the query to use to find the pre-existing record that this Input event updates.
boolean1	Boolean	No	A logical parameter flag, *true* or *false*, that indicates whether the eventout record writes on transaction completion.
text	character	No	Designates the output event type.

RAD routine: axces.software

A RAD routine that adds, updates, or deletes a software configuration item (CI) that an external agent discovers to the pcfiles file once the filter criteria are satisfied.

Routine

axces.software

Parameters

The following parameters are valid for the axces.software routine:

Name	Data type	Required	Description
record	record	No	Writes the unique record identifier to the eventin record. Set up the *$axces* variable using Format Control to create the intended result.
prompt	character	No	Indicates the Map name used in event registration.
string1	character	No	Identifies the software file in Service Manager where the configuration item (CI) data is recorded.
query	character	No	In this case, *logical.name=1* in $axces.fields and *description=9* in $axces.fields.
boolean1	Boolean	No	A logical parameter flag, *true* or *false*, that indicates whether the eventout record writes on transaction completion.
name	character	No	Specifies the Format Control record to use.

RAD routine: axces.write

A RAD routine used to build an eventout record used by the SCAutomate (SCAuto) interface.

Routine

axces.write

Parameters

The following parameters are valid for the axces.write routine:

Name	Data type	Required	Description
record	record	Yes	Names the record to be written. In Format Control the record is *$file*.
name	character	No	Names the registration type as it appears in the eventregister file. For example, to write an eventout record when an incident is opened, use the value *pmo* for this parameter.
string1	character	No	Defines the separation character. If you define your own character, ensure that it does not occur naturally in fields in the event. The default is value is the caret character (^).
text	character	No	Defines a system sequence ID with a maximum length of 16 characters. Service Manager generates the system sequence ID unless you supply one.

prompt	character	No	Defines a user sequence ID with a maximum length of 16 characters.
query	character	No	Defines the user name. The default value is *operator()*.

RAD routine: database

A RAD routine used to access other tables from a particular record. For example, to enable an option button to appear on your users' screens, allowing them to call the location table from the contacts table.

Note: This routine is only called from Additional Options in Format Control.

Routine
database

Parameters
The following parameter is valid for the database routine:

Name	Data type	Required	Description
name	character	No	The name of the form you want to open.
record	record	No	Specifies the Format Control record to use.

RAD routine: es.approval

A RAD routine that processes approvals for the Request Management and Change Management applications.

Routine
es.approval

Parameters
The following parameters are valid for the es.approval routine:

Name	Data type	Required	Description
record	record	Yes	Identifies the eventin record to pass.
text	character	No	Identifies the Service Manager table where data is recorded.
name	character	No	Indicates the Map name used in event registration.
cond.input	Boolean	No	A logical parameter, *true* or *false*, that specifies whether the eventout record writes on transaction completion.

RAD routine: fill.fc

A RAD routine that copies data from a target file to the current file using predefined links under user control.

Routine
fill.fc

Parameters
The following parameters are valid for the fill.fc routine:

Name	Data type	Required	Description
record	record	Yes	The value is $*file* when calling fill.fc through the Additional Options process.
text	character	No	The name of the field in the target file to fill. The default is the current field (the field where the cursor is located). If the field is an array, the fill operation attempts to copy data back to that item in the selected array.
string1	character	No	The form name that determines which link record to use. The default is the link record for the current form.

Variables supported
$fill.replace
$project.first
$fill.exact

RAD routine: getnumb.fc

A RAD routine used to establish the numbering sequence in sequential numbering. You can create simple identification numbers for database records, or complex numbers composed of prefixes and suffixes.

Routine
getnumb.fc

Parameters
The following parameters are valid for the getnumb.fc routine:

Name	Data type	Required	Description
record	record	Yes	The data record where the number will be placed. The value of this parameter must be *$file*.
name	character	Yes	The sequential number Class (category) of the record that the Format Control record is attached. This value appears in the Class field of the Sequential Numbering File.

prompt	character	Yes	Names the field in the data record where the sequential number appears.
text	character	No	Defines the data type of the sequential number (number or string).
number1	number	No	Defines the length of the sequential number. The subroutine prefixes as many zeroes to the number as necessary to match the required number of digits. Pass the RAD function val() to the number1 parameter. For example, a four-digit number has a value of *val("4", 1)*. Where: The number *4* is the length of the number and the number *1* is the data type (a number representation).
string1	character	No	Defines the prefix. This is the identifying character string that appears before the number.
numbers, 1	array (numbers)	No	Determines the reset point for the sequential number. When the value passed to the application is reached, the sequential number resets to the starting value.
numbers, 2	array (numbers)	No	Establishes a new starting value for the sequential number when the reset point is reached.
numbers, 3	array (numbers)	No	Sets the increment or decrement value. For example, if you want to increment by 2, displaying odd or even numbers only.
numbers, 4	array (numbers)	No	Allows the user to increment or decrement the sequential number independent of the number file. Use this parameter to attach a specified number to your records without updating the number file.

RAD routine: marquee.publish

A RAD routine used to send a marquee message. This routine is always called from within marquee.send.

Routine

marquee.publish

Parameters

The following parameters are valid for the marquee.publish routine:

Name	Data type	Required	Description
name	character	Yes	The published variable used by the agent to search the

			database. **Note**: The system automatically adds *$MARQUEE* to the beginning your variable name before publishing it.
text	character	No	A variable that contains the message to be published.
prompt	character	No	The display color for your marquee or the valid system number in the Color field. The default value is Red.
boolean1	Boolean	No	A logical field. Set the flag to *true* to add this message to the marquee table. The marquee agent then reads the record and publishes it. If the flag is set to *false*, the routine immediately publishes your message without adding it to the marquee table (recommended method).
cond.input	Boolean	No	A logical field. Set the flag to *true* to add the name to the database if that record does not exist.

RAD routine: marquee.send

A RAD routine used to read the schedule record and initiate the marquee.publish application.

Routine
marquee.send

Parameters
The following parameters are valid for the marquee.send routine:

Name	Data type	Required	Description
record	character	Yes	Schedule record for the marquee process
boolean1	Boolean	No	Logical field. Set the flag to *true* to send all marquee messages defined in the schedule record. If set to *false*, or the field is *NULL*, the application publishes only the specific marquee message defined in the schedule record (recommended method).

RAD routine: message.fc

A RAD routine used to send messages under user control.

Routine
message.fc

Parameters

The following parameters are valid for the message.fc routine:

Name	Data type	Required	Description
index	number	No	Select one of three message levels: • *1* for information, this is the default. • *2* for action. • *3* for error. In Text mode, message levels may appear in different colors.
prompt	character	No	Enter a message class that matches one of the records in the msgclass table, such as problemclose. To send email messages, there must be a msgclass record with a type of email for the message class name specified. The default value is *msg*.
text	character	Yes	The text of the message can be a string or an array. For example, generate an array of the screen contents using the genout() function and then insert lines of text at the top of the array.
name	character	No	This value can contain either a list of operator names or a single name. For internal Service Manager messages, the names must be operator ID's defined in the operator table. For email, the names can either be operator ID's or contact names (contact.name) defined in the contacts table; an email address must be specified in the relevant table. The default is operator().
string1	character	No	The message name parameter identifies the message. Use the name of the Service Manager application or application area that generates the message.
number1	number	No	The message number parameter identifies a message within the area specified by the string1 parameter.
query	character	No	The mail class parameter is used within Incident Management applications to identify the ticket number so that mail already sent can be selected and updated. It must contain the string pm.main and a Mail Target must be defined.

names,1	array (characters)	No	The mail target parameter must contain the incident record number.

RAD routine: post.fc

A RAD routine used to execute the posting routine from the subroutines process (automatic posting) and from the additional options process (manual posting).

Routine
post.fc

Parameters
The following parameters are valid for the post.fc routine:

Name	Data type	Required	Description
file	record	Yes	The value is always *$file* when calling post.fc through the Additional Options process. In most cases, the variable is *$file* when calling post.fc through the Subroutine process; however, it could be *$file1*, or another file variable established by Format Control or custom RAD processing.
name	character	No	Defaults to the name of the currently open form. You can override this default by passing a string of the specific Link record you want to use.
prompt	character	No	Defaults to the name of the input field that the cursor is in when the you select the option. You can override this default by passing a string of a specific field name you want to use.

RAD routine: publish

A RAD routine used to publish a variable into SYSPUB.

Routine
publish

Parameters
The following parameters are valid for the publish routine:

Name	Data type	Required	Description
name	character	No	The published variable used by the agent to search the database.
text	character	No	The variable that identifies the message to be published.

RAD routine: query.stored

A RAD routine used to access all queries for a specific table or to execute a specific stored query.

Note: This routine is called only from Additional Options in Format Control.

Routine
query.stored

Parameters
The following parameters are valid for the query.stored routine:

Name	Data type	Required	Description
name	character	No	The name of the table you want to enable the user to query, for example, *device.workstation*.
text	character	No	The name of the stored query you want to execute, for example, *name=operator()*.

RAD routine: scauto.inventory

A RAD routine that provides access to Configuration items (CIs).

Routine
scauto.inventory

Parameters
The following parameters are valid for the scauto.inventory routine:

Name	Data type	Required	Description
record	record	Yes	Writes the unique record identifier to the eventin record. If you use a variable such as *$axces*, set it up using Format Control to create the intended result.
prompt	character	No	Indicates the Map name in event registration.
string1	character	No	Identifies the Service Manager table name to write to.
text	character	No	Provides the specific action to take, such as add, update, or delete.
query	character	No	Indicates the query to use to find the pre-existing record that this Input event updates.
boolean1	Boolean	No	The logical parameter flag, *true* or *false*, that indicates whether to write the eventout record on transaction completion.
name	character	No	Specifies the event registration name.

RAD routine: sort.array

A RAD routine called from subroutines to sort simple arrays, such as number, character, or date and time, in either ascending or descending order.

Routine
sort.array

Parameters
The following parameters are valid for the sort.array routine:

Name	Data type	Required	Description
file	record	Yes	When calling sort.array from Format Control, always pass *$file* to this parameter.
name	character	Yes	The name of the input field (array) in $file you want to sort.
boolean1	Boolean	No	Controls the sort order. A value of *true* sorts in ascending order, and a value of *false*sorts in descending order. The value is expressed as *val()*, for example, *val ("false", 4)*.

RAD routine: validate.fields

A RAD routine used to validate fields in a form.

Routine
validate.fields

Parameters
The following parameters are valid for the validate.fields routine:

Name	Data type	Required	Description
name	character	No	Names the field in the current form to validate, such as logical.name. To name the field in which the cursor was last located when an add or update was performed, use the RAD function cursor.field.name().
names	array (of characters)	No	Name of the array in the current form containing the field names to validate. The line of the array to validate must be defined in the Initializations process.
second.file	record	Yes	The data record in which the field is being validated. The value of this parameter must be *$file*.

cond.input	Boolean	No	The required value of the field defined by the parameter val(). For example, to validate a Boolean field to *true*, use the value *val("true", 4)*. When set to true, the value can be looked up online. This means a list of valid values is presented for selection.

RAD routine: wmi.inventory.check

A RAD routine that checks the eventin record for status information from a type of scan. This routine also processes the information from the scan into the Configuration Management application using the scauto.inventory RAD routine.

Routine

wmi.inventory.check

Parameters

The following parameters are valid for the wmi.inventory.check routine:

Name	Data type	Required	Description
record	record	Yes	Identifies the eventin record to pass.
prompt	character	No	Indicates the Map name used in event registration.
string1	character	No	Identifies the Service Manager table where data is recorded.
text	character	No	Provides the specific action to take, such as add, update, or delete.
query	character	No	Indicates the query to use to find the pre-existing record that this Input event updates.
boolean1	Boolean	No	A logical parameter flag, *true* or *false*, that indicates whether the eventout record writes on transaction completion.
name	character	No	Identifies the event registration name.
cond.input	Boolean	No	A logical parameter, *true* or *false*, that specifies whether or not to process additional records if you select more than one record.

Chapter 6

JavaScript and Service Manager reference

This chapter provide the reference details for the JavaScript global System objects, JavaScript global methods, Service Manager defined JavaScript objects, JavaScript functions available in Service Manager, JavaScript functions available in the ScriptLibrary of Service Manager, and some JavaScript examples.

JavaScript global System objects

Global objects are predefined objects that are always instantiated and cannot be created using the “new” operator. Objects have methods and/or subordinate properties that you refer to by using the dot operator.

The primary global object is the **System** object. It acts as a container for an entire hierarchy of objects. These other objects are properties of the System object.

Note: None of the Service Manager global System objects have constructors.

This is a complete list of global objects exposed as properties of the System object.

- system.files - see "JavaScript object: system.files" on next page
- system.forms - see "JavaScript object: system.forms" on next page
- system.functions - see "JavaScript object: system.functions" on page 205
- system.library - see "JavaScript object: system.library" on page 207
- system.oldrecord - see "JavaScript object: system.oldrecord" on page 208
- system.record - "JavaScript object: system.record" on page 208
- system.sysinfo - "JavaScript object: system.sysinfo" on page 209
- system.threads - "JavaScript object: system.threads" on page 209
- system.user - "JavaScript object: system.user" on page 210
- system.users - "JavaScript object: system.users" on page 210
- system.vars - "JavaScript object: system.vars" on page 211

Note: All of the alias or shortcut global objects available in HP ServiceCenter 6.0 are deprecated in Service Manager 9.30. They are deprecated because the alias or shortcut is likely to be used as a script variable. The scripts that attempt to use these names as variables will not function as expected. Therefore, you must type out the entire global system object name. For example, type `system.user` instead of `user`.

JavaScript object: system.files

The following JavaScript object is unique to HP Service Manager.

The system.files object allows you to call a particular Service Manager table into memory.

Usage

`system.files.`*`table name`*

Note: Service Manager global system objects do not have constructors.

Arguments

There are no arguments for this object.

Properties

The following properties are valid for this object:

Property	Required	Description
table name	Yes	This argument contains the table name you want to bring into memory.

Methods

None

Example

This example does the following:

- Sets a variable equal to system.files.contacts
- Displays the contents of the contacts table

This example requires the following sample data:

- A valid Service Manager table name (for example, contacts)

```
var f = system.files.contacts;
print( "The value of system.files.contacts is:\n" + f );
```

JavaScript object: system.forms

The following JavaScript object is unique to HP Service Manager.

The system.forms object allows you to call a particular Service Manager form into memory.

Usage

`system.forms.`*`form name`*

Note: Service Manager global system objects do not have constructors.

Arguments

There are no arguments for this object.

Properties

The following properties are valid for this object:

Property	Required	Description
form name	Yes	This argument contains the form name you want to bring into memory.

Methods

None

Example

This example does the following:

- Sets a variable equal to system.forms.ScriptLibrary
- Displays the contents of the ScriptLibrary form

This example requires the following sample data:

- A valid Service Manager form name (for example, ScriptLibrary)

```
var fm = system.forms.ScriptLibrary;
print( "The value of system.forms.ScriptLibrary is:\n" + fm );
```

JavaScript object: system.functions

This JavaScript object is unique to HP Service Manager. The system.functions object allows you to call a particular Service Manager RAD function from JavaScript.

Usage

`system.functions.`*`RAD function name`*`(` *`RAD function arguments`* `)`

Note: There are two exceptions to syntax used for RAD functions: *delete* and *null*

- `system.functions.`*`_delete`*`(` *`RAD function arguments`* `)`
- `system.functions.`*`_null`*`(` *`RAD function arguments`* `)`

Note: Service Manager global system objects do not have constructors.

Arguments

The following arguments are valid for this object:

Argument	Data type	Required	Description
RAD function arguments	String	Yes	This argument contains any arguments required to execute the RAD function.

Properties

The following properties are valid for this object:

Property	Required	Description
RAD function name	Yes	This argument contains the RAD function name you want to bring into memory.

Methods

None

Example

This example does the following:

- Sets a variable equal to system.functions.operator()
- Displays the results of the RAD operator() function

This example requires the following sample data:

- A valid Service Manager RAD function name (for example, operator())

```
var operator = system.functions.operator();
print( "The value of system.functions.operator() is:\n" + operator
);
```

JavaScript object: system.functions.scmsg

The following JavaScript object is unique to HP Service Manager.

The system.functions.scmsg object allows you to call a particular Service Manager message record from JavaScript.

Usage

`system.functions.scmsg( message number, message class, message elements )`

Note: Service Manager global system objects do not have constructors.

Arguments

The following arguments are valid for this object:

Argument	Data type	Required	Description
message number	String	Yes	This argument contains the Service Manager message number you want to call into memory.
message class	String	Yes	This argument contains the Service Manager message class you want to call into memory.
message elements	Array	No	This argument contains the variable message elements indicated by %S in the message record.

Properties

None

Methods

None

Return values

A string containing the complete message text indicated by the arguments.

Example

This example does the following:

- Sets variables to define the message record
- Calls the RAD function scmsg()
- Displays the results of the RAD function scmsg()

This example requires the following sample data:

- A valid Service Manager message record (for example, "km" class record 14 "Group %S and Profile %S are already in category %S")

```
var msgrecord = "14";
var msgclass = "km";
var arguments = new Array();
arguments.push( "New Group" );
arguments.push( "New Profile Name" );
arguments.push( "Category Name" );
print( system.functions.scmsg( msgrecord, msgclass, arguments ) );
```

JavaScript object: system.library

The following JavaScript object is unique to HP Service Manager.

The system.library object allows you to call a particular Service Manager script library function.

Usage

`system.library.`*`script name`*`.`*`function name`*`(` *`function arguments`* `)`

Note: Service Manager global system objects do not have constructors.

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
function arguments	String	Yes	This argument contains the function arguments you want to use.

Properties

The following properties are valid for this object:

Property	Required	Description
script name	Yes	This argument contains the script name you want to run.
function name	No	This argument contains the function name you want to run.

Methods

None

Example

This example does the following:

- Creates an array
- Calls a script to check if a value is in the array

```
var testArray = new Array();
testArray.push("Blue");
testArray.push("Pink");
var result = system.library.ArrayUtil.contains(testArray, "Blue");
```

```
print("The value you specified is in the array:  " + result);
```

JavaScript object: system.oldrecord

The following JavaScript object is unique to HP Service Manager.

The system.oldrecord object allows you to call the last Service Manager record into memory.

Usage

`system.oldrecord`

Note: Service Manager global system objects do not have constructors.

Arguments

There are no arguments for this object.

Properties

There are no properties for this object.

Methods

None

Example

This example does the following:

- Sets a variable equal to system.oldrecord
- Displays the results of system.oldrecord

```
var o = system.oldrecord;
print( "The value of system.record is:\n" + o );
```

JavaScript object: system.record

The following JavaScript object is unique to HP Service Manager.

The system.record object allows you to call the current Service Manager record into memory.

Usage

`system.record`

Note: Service Manager global system objects do not have constructors.

Arguments

There are no arguments for this object.

Properties

There are no properties for this object.

Methods

None

Example

This example does the following:

- Sets a variable equal to system.record
- Displays the results of system.record

```
r = system.record;
print( "The value of system.record is:\n" + r );
```

JavaScript object: system.sysinfo

The following JavaScript object is unique to HP Service Manager.

The system.sysinfo object allows you to call an XML list of properties about the Service Manager server.

Usage

```
system.sysinfo
```

Note: Service Manager global system objects do not have constructors.

Arguments

There are no arguments for this object.

Properties

There are no properties for this object.

Methods

None

Example

This example does the following:

- Sets a variable equal to system.sysinfo
- Displays the results of system.sysinfo

```
var s = system.sysinfo;
print( "The value of system.threads is:\n" + s );
```

JavaScript object: system.threads

The following JavaScript object is unique to HP Service Manager.

The system.threads object allows you to call an XML list of the current Service Manager threads into memory.

Usage

```
system.threads
```

Note: Service Manager global system objects do not have constructors.

Arguments

There are no arguments for this object.

Properties

There are no properties for this object.

Methods
None

Example
This example does the following:

- Sets a variable equal to system.threads
- Displays the results of system.threads

```
var th = system.threads;
print( "The value of system.threads is:\n" + th );
```

JavaScript object: system.user

The following JavaScript object is unique to HP Service Manager.

The system.user object allows you to call an XML list of properties of the currently logged on Service Manager user.

Usage
`system.user`
Note: Service Manager global system objects do not have constructors.

Arguments
There are no arguments for this object.

Properties
There are no properties for this object.

Methods
None

Example
This example does the following:

- Sets a variable equal to system.user
- Displays the results of system.user

```
var u = system.user;
print( "The value of system.user is:\n" + u );
```

JavaScript object: system.users

The following JavaScript object is unique to HP Service Manager.

The system.users object allows you to call an XML list of the currently logged on Service Manager users into memory.

Usage
`system.users`
Note: Service Manager global system objects do not have constructors.

Arguments
There are no arguments for this object.

Properties
There are no properties for this object.

Methods
None

Example
This example does the following:

- Sets a variable equal to system.users
- Displays the results of system.users

```
u = system.users;
print( "The value of system.users is:\n" + u );
```

JavaScript object: system.vars

The system.vars object is unique to HP Service Manager.

The system.vars object allows you to call a particular Service Manager variable into memory.

Usage
`system.vars.`*`variable name`*
Note: Service Manager global system objects do not have constructors.

Arguments
There are no arguments for this object.

Properties
The following properties are valid for this object:

Property	Required	Description
variable name	Yes	This argument contains the variable name you want to bring into memory.

Methods
None

Example
This example does the following:

- Sets a variable equal to system.vars.$L_file
- Displays the contents of the $L_file variable

This example requires the following sample data:

- A valid Service Manager variable name (for example, $L_file)

```
var v = system.vars.$L_file;
print( "The value of system.vars.$L_file is:\n" + v );
```

List: JavaScript global methods

Global methods are functions that are available to any script as they are not methods of any specific object. You can invoke global methods directly just as you would do with any core JavaScript global functions such as parseInt() or eval().

This is a complete list of all the global methods available in HP Service Manager.

Global method	Description
base64Decode	Converts base 64 string data to its original format.
base64Encode	Converts binary data to a base 64 string format.
compile	Validates the syntax of the specified JavaScript.
doHTTPRequest	Issues an HTTP request to a specified URL.
doSOAPRequest	Issues an SOAP request to a specified URL.
execute	This method performs the specified process or program.
help	Displays a brief description of a Service Manager-defined JavaScript object.
makeSCWebURL	Creates a URL query to the Service Manager Web tier.
print	Displays a message in the client Messages view.
Quit	Allows JavaScript to abort processing and return a failure return code.
RCtoString	Converts a Service Manager global return code value into a localized text string.
readFile	Reads data from the local file system.
setAppMessage	Defines the message returned in the "message" attribute in the soap response.
stripHtml	Takes HTML content and strips out the HTML tags and returns the content as text without the HTML tags.
uncompressFile	Expands a .zip file into a specified location.
writeAttachmentToFile	Writes a requested attachment record to the local file system.
writeFile	Writes data to the local file system.
xmlstring	Converts a JavaScript string to an XML string.

JavaScript global method: base64Decode

Converts base 64 string data to its original format.

Syntax

```
base64Decode( data );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
data	Binary	Yes	This argument contains the data you want the script to decode to its original format.

Return values

A binary or string object containing the original data.

Description

This method converts data in base 64 string format to its original format.

Example

This example does the following:

- Reads the contents of a binary file
- Converts the file to a base 64 string format
- Writes the base 64 string data to a local file
- Prints the number of bytes written to the local file system
- Reads the contents of a base 64 string format file
- Converts the data into a binary file
- Writes the binary data to a local file
- Prints the number of bytes written to the local file system

This example requires the following sample data:

- A binary file on the local file system (for example, an image file)

```
var source = readFile( "C:\\header_left.gif", "b" );
var encode = base64Encode( source );
var encodedFile = writeFile( "C:\\base64EncodedImage.txt", "b",
encode );
print( "Wrote " + encodedFile + " bytes to
C:\\base64EncodedImage.txt" );
source = readFile( "C:\\base64EncodedImage.txt", "t" );
var decode = base64Decode( source );
var decodedFile = writeFile( "C:\\newImage.gif", "b", decode );
print( "Wrote " + decodedFile + " bytes to C:\\newImage.gif" );
```

JavaScript global method: base64Encode

Converts binary data to a base 64 string format.

Syntax

```
base64Encode( data );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
data	Binary	Yes	This argument contains the data you want the script to encode in base 64 format.

Return values

A string object containing the data in base 64 format.

Description

This method converts binary data to a base 64 string format. You can also use this method to encode plain text data in base 64 format.

Example

This example does the following:

- Reads the contents of a binary file
- Converts the file to a base 64 string format
- Prints the base 64 string

This example requires the following sample data:

- A binary file on the local file system (for example, an image file)

```
var source = readFile( "C:\\header_left.gif", "b" );
var encode = base64Encode( source );
print( "The value of encode is:\n" + encode );
```

JavaScript global method: compile

Validates the syntax of the specified JavaScript.

Syntax

```
compile( script, description );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
script	String	Yes	This argument contains the name of the script you want to validate.
description	String	No	Use this argument to store any comments you want.

Return values

A success or error message.

This method displays the message "Successful compilation of JavaScript function or expression" if the JavaScript passes validation.

Throws

None

Description

This method validates the syntax of a JavaScript and displays a message as to the success or failure of the validation. This method is an alias of the execute() method.

Example

This example does the following:

- Creates a variable to store a JavaScript name
- Validates the contents of the script

This example requires the following sample data:

- A valid JavaScript

```
var s = "lib.SCFILEgetTextTest"
compile( s, "Testing execute" );
```

JavaScript global method: doHTTPRequest

Issues an HTTP request to a specified URL.

Syntax

```
doHTTPRequest (HTTP command, URL, Headers, and POST body)
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
HTTP command	String	Yes	This argument specifies a command verb. We support GET, POST, PUT, HEAD, and DELETE.
URL	String	Yes	This argument specifies the URL to which the script should send an HTTP request. If the HTTP request requires SSL then the URL must start with https.
Headers	Array	Yes	This argument contains a JavaScript array of Service Manager-defined Header objects or an empty array. See "JavaScript object: Header" on page 236 for more information.
POST body	String	No	This argument contains the body of a POST command request.

Return values

A string containing an HTTP response or an error message.

Description

This method sends an HTTP request to a remote server.

Example

This example does the following:

- Sends an HTTP request to the Service Manager Web services API
- If successful, writes the response to a local file
- If unsuccessful, displays the HTTP error message

Note: To see a working example of the doHTTPRequest method that incorporates multiple error checking conditions, open the SOAP script in the script library. The WSDL2js function uses the doHTTPRequest method at line 114 as part of a try/catch statement.

This example requires the following sample data:

- The URL to the Service Manager Web services API

```
// Use the following url value to produce an HTTP 400 error.
//var url = "http://localhost:13080/IncidentManagement.wsdl";

// Use the following url value to produce valid WSDL.
var url = "http://localhost:13080/SM/7/IncidentManagement.wsdl";

var headers = new Array();
var WSDLrequest;
var reply;
try
       {
        WSDLrequest = doHTTPRequest( "GET", url, headers, null, 10, 10 ,10
);
        print( "The result of the doHTTPRequest is \n" + WSDLrequest );
        reply = writeFile( "C:\\IncidentManagement.wsdl", "text",
WSDLrequest );
       }
catch ( e )
       {
        print( "WSDL request failed with exception \n" + e );
       }
```

JavaScript global method: doSOAPRequest

Issues a SOAP request to a specified URL.

Syntax

```
doSOAPRequest( URL, SOAPAction, XML, SOAP user ID, SOAP password,
Connect Timeout, Read Timeout, Obsolete, Attachment object,
AcceptFastInfoset,);
```

Parameters

The following arguments are valid for this function:

Argument	Data type	Required	Description
URL	String	Yes	This argument specifies the URL to which the script should send an HTTP request. If the

			HTTP request requires SSL then the URL must start with https.
SOAPAction	String	Yes	This argument specifies the SOAPAction value.
XML	String	Yes	This argument contains the XML for an entire SOAP request including the SOAP envelope. Refer to the Web Service's WSDL to determine the proper format of the SOAP request.
SOAP user ID	String	No	This argument contains the user ID to be used for the Basic Authorization HTTP header.
SOAP password	String	No	This argument contains the password value to be used for the Basic Authorization HTTP header.
Connect Timeout	Number	No	This argument specifies the number of seconds a SOAP request has to establish a connection before it times out.
Read Timeout	Number	No	This argument specifies the number of seconds a SOAP request has to read data before it times out.
Obsolete	Number	No	This argument is obsolete.
Attachment object	Array	No	This argument contains a JavaScript array of Service Manager-defined Attachment objects or an empty array. See "JavaScript object: Attachment" on page 232 for more information.
AcceptFastInfoset	Number	No	To set the SOAP Message flag "acceptfastinfoset". Default value is 1. To turn "acceptfastinfoset" off: 0.

Return values

A string containing a SOAP response or an error message.

Description

This method sends a SOAP request to a remote server and returns the SOAP response.

Examples

Example 1 :

- Sends a SOAP request to a delayed stock quote Web service and indicates SM accept the "fastinfoset" formatted SOAP message
- Prints the SOAP response to the screen

This example requires the following sample data:

- The URL to a delayed stock quote Web service
- Stock symbol for company

```
var url =
"http://ws.cdyne.com/delayedstockquote/delayedstockquote.asmx";
var action = "http://ws.cdyne.com/GetQuote";
var xml = "<soap:Envelope
xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\"

    xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"

    xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"

    xmlns:s0=\"http://ws.cdyne.com/\">

    <soap:Body>

           <GetQuote xmlns=\"http://ws.cdyne.com/\">

           <StockSymbol xsi:type=\"s:string\">GOOG</StockSymbol>

           <LicenseKey xsi:type=\"s:string\">0</LicenseKey>

           </GetQuote>

    </soap:Body></soap:Envelope>";
var uid = null;
var pass = null;
var attachments = new Array();
var quote = doSOAPRequest( url, action, xml, uid, pass, 60, 60,
null, attachments );
print( "The return value is: " + quote );
```

Example 2:

- Sends a SOAP request to a delayed stock quote Web service and indicates SM does not accept the "fastinfoset" formatted SOAP message.
- Prints the SOAP response to the screen

This example requires the following sample data:

- The URL to a delayed stock quote Web service
- Stock symbol for company

```
var url =
"http://ws.cdyne.com/delayedstockquote/delayedstockquote.asmx";
var action = "http://ws.cdyne.com/GetQuote";
var xml = "<soap:Envelope
xmlns:soap=\"http://schemas.xmlsoap.org/soap/envelope/\"

    xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"

    xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"

    xmlns:s0=\"http://ws.cdyne.com/\">

    <soap:Body>

        <GetQuote xmlns=\"http://ws.cdyne.com/\">

        <StockSymbol xsi:type=\"s:string\">GOOG</StockSymbol>

        <LicenseKey xsi:type=\"s:string\">0</LicenseKey>
```

```
        </GetQuote>
    </soap:Body></soap:Envelope>";
var uid = null;
var pass = null;
var attachments = new Array();
var quote = doSOAPRequest( url, action, xml, uid, pass, 60, 60,
null, attachments, 0 );
print( "The return value is: " + quote );
```

JavaScript global method: execute

Validates the syntax of the specified JavaScript, and then runs it.

Syntax

```
execute( script, description );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
script	String	Yes	This argument contains the name of the script you want to validate.
description	String	No	Use this argument to store any comments you want.

Return values

A success or error message.
This method displays the message "Successful compilation of JavaScript function or expression" if the JavaScript passes validation.

Throws

None

Description

This method validates the syntax of a JavaScript, and then runs it. The method also displays a message as to the success or failure of the validation.

Example

This example does the following:

- Creates a variable to store a JavaScript name
- Validates the contents of the script
- Runs the script

This example requires the following sample data:

- A valid JavaScript

```
var s = "lib.SCFILEgetTextTest"
execute( s, "Testing execute" );
```

JavaScript global method: help

Displays a brief description of a Service Manager-defined JavaScript object.

Syntax

```
help( object );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
object	Service Manager-defined JavaScript object	Yes	This argument contains the Service Manager-defined JavaScript object for which you want a brief description.

Return values

A string or null.

This method returns a brief text string describing the Service Manager-defined JavaScript object or null if there is no help available.

Throws

None

Description

This method displays a brief description of a Service Manager-defined JavaScript object. You must use the print() method to see the help text string.

Example

This example displays the help contents of several Service Manager-defined JavaScript objects.

```
var f = new SCFile();
print( help( f ) );
var x = new XML();
print( help( x ) );
var d = new XMLDate();
print( help( d ) );
```

JavaScript global method: makeSCWebURL

Creates a URL query to the Service Manager Web tier.

Syntax

```
makeSCWebURL( Web tier URL, docEngine, table name, query, hash key
seed, action);
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description

Web tier URL	String	Yes	This argument specifies the fully qualified URL to the Service Manager Web tier. This URL must include the http:// protocol syntax as well as the server name or IP address and communications port number.
docEngine	String	Yes	This argument specifies that the query should be handled by the Service Manager Document Engine. This argument does not accept any other string value.
table name	String	Yes	This argument specifies the table where the Document Engine should query for records.
query	String	Yes	This argument specifies the Service Manager query you want to use to search for records. You must URI encode the query string to prevent special characters from invalidating the URL.
hash key seed	String	No	This argument specifies an optional string you want to use to create a unique hash key. The hash key seed does not appear in the final URL produced.
action	String	Not applicable	This argument cannot be used. By default, the URL query always performs a search operation.

Return values

A string containing a URL query to the Service Manager Web tier.

Description

This method creates a URL query to the Service Manager Web tier. It does not convert any special characters in the URL (such as spaces or quotation marks) to the character code equivalent in URI format. You can use the encodeURIComponent method to convert any special characters in the URL.

Example

This example does the following:

- Creates a URL query to the Service Manager Web tier
- Converts any special characters in the URL to valid URI format

This example requires the following sample data:

- The URL to Service Manager Web tier
- A valid Service Manager table name (for example, incidents)
- A valid Service Manager query against the table (for example, incident.id="SD1001")

```
function createURLquery( table, query)
{

 var url;
 var webtier;
 var webserver;
 var doceng;
```

```
 var hashkey;
 var action;

 print( "Parameters: table="
        + table +
        "query='"
        + query + "'");
 print( "Converting special characters in query to valid URI
format..." );
 queryURI = encodeURIComponent( query );
 print( "Converted: old query='"
        + query +
        "' new query='"
        + queryURI +
        "'" );

 webserver = "http://pdoref02/sc/index.do";
 doceng = "docEngine";
 hashkey = "";
 action = "";
 webtier = makeSCWebURL( webserver,
           doceng,
           table,
           queryURI,
           hashkey,
           action, "abc");
 url=encodeURIComponent(webtier);
 print( "Creating URL from information provided..." );
 print( "The value of webtier is:\n" + webtier );
 print( "The value of url is:\n" + url );
 return url;
}

var tablename = "incidents"
var SCquery = "incident.id=\"SD1001\""

createURLquery( tablename, SCquery);
```

JavaScript global method: print

Displays a message in the client Messages view and prints it in the Service Manager log file.

Note: Messages only print to the Service Manager log file when JavaScript is running in the background.

Syntax

```
print( output );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
output	String	Yes	This argument contains the text string or variable value you want the script to print to the Message View and the sm.log file. You must enclose literal text strings in quotation marks.

Return values

None

Throws

None

Description

This method displays the contents of the method argument in the Messages view and also prints it in the Service Manager log file.

Example

This example does the following:

- Creates a variable with a string value
- Displays the contents of the variable

This example requires the following sample data:

- A text string

```
var example = "abc123";
print("The value of example is: " + example);
```

JavaScript global method: Quit

Allows JavaScript to abort processing and return a failure return code.

Syntax

```
Quit( return code );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
return code	Integer	Yes	This argument contains the numeric return code you want the script to return upon ending.

Return values

A failure return code

Throws

None

Description

This method stops the processing of JavaScript from the point it is called and returns the failure return code specified in the *return code* argument. Any JavaScript code after the Quit() method is ignored.

Example

This example does the following:

- Prints a message
- Quits and returns a failure return code

```
print( "Testing the Quit() method..." );
Quit( -1 );
print( "You won't see this message because the JavaScript has
already quit" );
```

JavaScript global method: RCtoString

Converts a Service Manager global return code value into a localized text string.

Syntax

```
RCtoString( return code );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
return code	Integer	Yes	This argument contains the Service Manager global return code value you want to convert to a localized string.

Return values

A localized text string

Throws

None

Description

This method converts a Service Manager global return code value into a localized text string.

Example

This example does the following:

- Searches the contacts table for any contact name you define in the search variable
- Displays the contact record as a text string

This example requires the following sample data:

- A valid contact name (for example, "ADMINISTRATOR, SYSTEM")
- A invalid contact name (for example, "NOT A, CONTACT")

```
var contactName;
```

```
function findContactName( name )
{
 print( "Searching for contact: " + name + "..." );
 var contactList = new SCFile( "contacts" );
 var findContact = contactList.doSelect( "contact.name=\""+ name +
"\"" )
 if ( findContact == RC_SUCCESS )
 {
  print( "Success. found " + name + " in contact record:\n" +
contactList.getText() );
  return contactList
 }
 else
 {
  print( "Could not find contact. " + RCtoString( findContact ) );
  return null
 }
}

contactName = "ADMINISTRATOR, SYSTEM";
findContactName( contactName );

contactName = "NOT A, CONTACT";
findContactName( contactName );
```

JavaScript global method: readFile

Reads data from the file system on the Service Manager server host.

Syntax

```
readFile( path, binary );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
path	String	Yes	This parameter contains the text string or variable value containing the fully qualified path of the file to be read. You must enclose literal text strings in quotation marks.
binary	String	Yes	This parameter determines if the file format is binary or text. Use the string "b" or "B" to read a binary file. Use any other value to read a text file.

Return values

A string object containing the contents of the file.

Description

This method reads the contents of a local file and returns a string object. The file can be in binary format or text format as determined by the binary argument.

Example

This example does the following:

- Reads the contents of a text file
- Reads the contents of a binary file

This example requires the following sample data:

- A text file on the local file system
- A binary file on the local file system

```
var textFile = readFile( "C:\\test.xml", "t" );
print( "The value of textFile is: " + textFile );

var binFile = readFile( "C:\\header_left.gif", "b" );
print( "The value of binFile is: " + binFile );
```

JavaScript global method: setAppMessage

Defines the message returned in the "message" attribute in the soap response. For example, in the following soap response:

<CreateRelationshipResponse *message*="Here you define your message" returnCode=….>, where the *message* attribute is shown in italic .

Syntax

```
setAppMessage(message);
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
message	String	Yes	This argument contains the text string you want to show in the "message" attribute in the soap response. You must enclose literal text strings in quotation marks.

Return values

None

Throws

None

Description

This method defines the message returned in the "message" attribute in the soap response.

Example

```
inputValue = "CompanyA";
setAppMessage("The input value "+inputValue+" is invalid");
```

JavaScript global method: stripHtml

Takes HTML content and strips out the HTML tags and returns the content as text without the HTML tags.

Syntax

```
stripHtml(htmlstr);
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
htmlstr	String	Yes	This argument contains the JavaScript String with HTML tags that you want to strip out of the string.

Returns

JavaScript String without HTML tags.

Throws

None

Description

Takes HTML content and strips out the HTML tags and returns the content as text without HTML tags.

Example

```
var htmlstr = "<html><body>Hello</body></html>";
var text = stripHtml(htmlstr);
print("before stripping HTML tags: " + htmlstr);
print("after stripping HTML tags: " + text);
```

JavaScript global method: uncompressFile

This method expands a .zip file into a specified location.

Syntax

```
uncompressFile(file name, target directory);
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
file name	String	Yes	This argument specifies the path and filename of the .zip file to uncompress.
target directory	String	No	This argument specifies the target directory. If not specified, the location of the .zip file is used.

Return values

True indicates that the file uncompressed successfully, and any other return value indicates failure.

Throws
None

Description
This method expands a .zip file into a specified location.

Example
This example unzips the file upgtest.zip to c:/test.

```
var rc = uncompressFile("c:/test/upgtest.zip");
if( rc == true )
{
  print("uncompress file with one file successfully");
  // this will unzip the file to C:/test/test
  rc = uncompressFile("c:/test/upgtest.zip", "c:/test/test");
}
else
{
print("Failed to uncompress file.");
}
if(rc == true)
{
  print("All tasks are finished.");
}
```

JavaScript global method: writeAttachmentToFile

Writes a requested attachment record to the local file system.

Syntax
`writeAttachmentToFile( File, Application, Topic, UID );`

Arguments
The following arguments are valid for this method:

Name	Data type	Required	Description
File	String	Yes	This argument specifies the file name of the attachment you want to write out to the file system. You may also use this argument to specify the destination path of the attachment. If this argument does not specify a path, Service Manager creates the attachment in the system working directory.
Application	String	Yes	This argument specifies the application name portion of the key that Service Manager uses to identify the attachment in the SYSATTACHMENTS table.
Topic	String	Yes	This argument specifies the topic name portion of the

			key that Service Manager uses to identify the attachment in the SYSATTACHMENTS table.
UID	String	Yes	This argument specifies the unique identifier portion of the key that Service Manager uses to uniquely identify the attachment in the SYSATTACHMENTS table.

Return values

RC_SUCCESS or one of the other global return code values.

The method writes the attachment record specified in the method arguments to the local file system and returns a global return code value of RC_SUCCESS or returns one of the failure global return code values if the method cannot return an attachment record.

Description

This method extracts the specified attachment from the SYSATTACHMENTS table and writes it to the specified file and path on the local file system.

Note: If you specify a path, you must escape out any JavaScript reserved-characters in the path name such as slashes and backslashes. Consult a JavaScript reference for a full list of JavaScript reserved-characters and escape methods.

Example

This example does the following:

- Requests an attachment from the SYSATTACHMENTS table
- Writes the attachment to the file c:\kmdocument.pdf
- Prints the return code to the screen

This example requires the following sample data:

- The Knowledge Management record KM0018
- An attachment in KM0018

```
var rc;
rc = writeAttachmentToFile( "c:\\kmdocument.pdf", "kmdocument",
"KM0018", "4462590a062ee0c2101d85c8" );
if ( rc == RC_SUCCESS )
{
 print( "kmdocument.pdf successfully created" );
}
else
{
 print( "kmdocument.pdf failed " , RCtoString(rc) );
}
```

JavaScript global method: writeFile

Writes data to the local file system.

Syntax

```
writeFile( path, mode, object );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
path	String	Yes	This argument contains the text string or variable value containing the fully qualified path of the file to be written. You must enclose literal text strings in quotation marks.
mode	String	No	Use the string "b" to write a binary file. Use the string "ba" to append to a binary file. Use the string "a" to append to a text file. Use any other string or omit the parameter to write to a text file. This is the default. The values are case sensitive and only the values specified are valid.
object	String	Yes	This parameter contains the text string or variable value containing the data you want to write to a file.

Return values

A number representing the number of bytes written to a file.

Description

This method writes the contents of the object parameter to the file specified in the path parameter. The file can be in binary format or text format as determined by the binary parameter.

Example

This example does the following:

- Writes the list of currently logged on users to a file

This example requires the following sample data:

- The list of currently logged on users stored in system.users

```
var userList = system.users;
var filePath;
var isMode;
var fileObject;

function writeToFile( path, binary, object )
{
 print( "Writing " + path + " to file..." );
 var output = writeFile( path, mode, object );
 print( "The number of bytes written to file was: " + output );
 return output
}

filePath = "C:\\users.xml";
isMode = null;
fileObject = userList;
```

```
writeToFile( filePath, isMode, fileObject );
```

JavaScript global method: xmlstring

This method converts a JavaScript string to an XML-formatted string.

Syntax

```
xmlstring( string );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
string	String	No	This argument contains the text string or variable value you want the script to convert to an XML string.

Return values

None

Throws

None

Example

```
var xmldoc = new SCFile( system.users );
xmlstring( xmldoc );
print( xmldoc );
```

Service Manager defined JavaScript objects

The defined objects are unique to the implementation of JavaScript within Service Manager. This is a complete list of the Service Manager defined objects.

Defined object	Description
Attachment	This object allows you to create attachments for use with doSOAPRequest methods.
Datum	This object extracts aggregate types of data (structures or arrays) from a record in the file object. In Service Manager, a file record is a datum and so is each component of the record.
SCDatum	This object converts a Datum to a SCDatum so that you can test the results by using the return codes. When an object is initialized in RAD and then passed to JavaScript, the object becomes a Datum object and all the methods return true/false values. If an error occurs and you want a more specific error message, you can create a SCDatum object from the Datum object.
SCFile	This object constructs a new file object. You can use the SCFile object to

	access, update, and move between Service Manager files. This object exposes a small set of frequently used return code constants that make it easy to test values returned by methods without having to hard code return code values in the script.
SCRecordList	This object is an array of Datums that qualify with the query. SCRecordList has two available functions: getPosition() and getCount().
Query	This object returns an array of records. The Query object can not be used to write data back to Service Manager.
QueryCond	This object defines the conditions of the query. It is used in conjunction with the Query object to define the parameters for returning data from Service Manager. The QueryCond object is similar to setConst method.
XML	This object represents an XML document or a node in an XML document. Underlying the XML object is the Document Object Model (DOM) object. You may find that in certain instances the XML object in Service Manager is not compliant with the DOM binding for JavaScript.
XMLDate	This object makes it possible to do conversions between three kinds of datetime values. These include the following constructors: XMLDate(dateobject); – it passes a JavaScript Date object to the constructor. XMLDate(iso8601datetimeordurationstring); – it passes an XML schema date, time, datetime, or duration string to the constructor. XMLDate(datetimedatum) – it passes a Service Manager datum object of type TIME to the constructor.

Note: In ServiceCenter version 6.1, the File and SCDoc objects are deprecated. HP strongly recommends that you do not use these objects since they may be removed in a future release.

JavaScript object: Attachment

The following JavaScript object is unique to HP Service Manager. This object allows you to create attachments for use with doSOAPRequest methods. You have to manually define the value property of each attachment object. For binary data attachments, you can use the readFile global method to assign the value property. If you need to create multiple attachments for a doSOAPRequest request, you can use the push method to add each Attachment object to a JavaScript array.

Constructor

```
new Attachment();
```

Arguments

None

Properties

The following properties are valid for this object:

Property	Data type	Description
value	Binary or String	Use this property to store the binary or string data of the attachment.
len	Integer	This property contains the file size of the attachment in bytes.
href	String	Use this property to store the unique identifier for the attachment in the SOAP request message.
action	String	Use this property to store the name of action to take with the attachment in the Service Manager Web Services API. The following options are available: • **add** – adds the specified attachment • **remove** – removes the specified attachment • **get** – retrieves the specified attachment • **update** – updates the specified attachment
name	String	Use this property to store the file name of the attachment.
type	String	Use this property to store the MIME type of the attachment.
attachmentType	String	Use this property to store the attachment type as defined in Service Manager. If present and the property has a value of "img," then the attachment must also have a MIME type of "image/gif".
stringValue	String	This is a read only property. When the attachment object contains a text value (utf-8 or ascii characters), user can use this property to get the string data of the attachment. The obtained string date then can be printed, concatenated, etc.

Methods

None

Example

This example does the following:

- Assume there are two attachments in the incident IM10005
- Get the existing attachments from the IM10005
- Concatenate these two attachments to a string
- Create a new attachment which contains the concatenated string value
- Insert this new attachment, named "MyAttachment.txt", into IM10005

This example requires the following sample data:

- A binary file (for example, an image)

```
var f = new SCFile( 'probsummary' );
var rc = f.doSelect( 'number = "IM10005"' );
```

```
var attachmentObj = f.getAttachments();
var str = "";
for ( var attachment in attachmentObj )
{
print("Attachment Name:  "+ attachmentObj[ attachment ].name );
print("Attachment Value:  "+ attachmentObj[ attachment ].stringValue
);
print("Attachment Value Length:  "+ attachmentObj[ attachment
].stringValue.length );
print("Attachment HREF:  "+ attachmentObj[ attachment ].href );
print("Attachment Type:  "+ attachmentObj[ attachment ].type);
str+= attachmentObj[ attachment ].stringValue ;
}
//Create new attachment with a concatenated value
print("String Length:  "+str.length);
var attachObj = new Attachment();
attachObj.type  = "text/plain";
attachObj.name  = "MyAttachment.txt";
attachObj.value = str;
var attachmentID = f.insertAttachment( attachObj );
print("AttachmentID:   "+attachmentID);
```

JavaScript object: Datum

The Datum JavaScript object is unique to HP Service Manager. This JavaScript object was deprecated as of HP ServiceCenter 6.1 because this object's methods can only return true or false boolean values. You can use the SCFile or SCDatum objects in place of this object to take advantage of the improved return code values.

With no parameters, the Datum() constructor creates an empty Datum object.

With an Object parameter, the Datum() constructor creates a Service Manager-formatted Datum object containing the provided record, structure, or array. You must enclose table names and arrays in quotation marks. You can use the Datum object as a parameter for RTECall routines that expect Service Manager-formatted parameters.

Constructor

```
new Datum();
new Datum(Object);
```

Arguments

The following arguments are valid for this object:

Argument	Data type	Description
Object	String	The Service Manager table name, record, structure, or array you want to query or update.

Properties

None

Methods

The following methods are valid for this object:

Defined method	Description
getText()	This method returns a text representation of the object.
getXML()	This method returns an XML representation of the object.
getSize()	This method returns the size of the object.
getMessages()	This method returns an array of messages generated by the Document Engine from a previous SCFile.doAction().
length()	This method returns the length of the object.
getNext()	This method moves to the next record if the object represents a result set.
getPrev()	This method moves to the previous record if the object represents a result set.
getFirst()	This method moves to the first record if the object represents a result set.
getLast()	This method moves to the last record if the object represents a result set.
doSelect()	This method makes a query. It no longer requires that the first argument be the name of the table if the object is constructed with the name of the table.
isRecord()	This method checks if an object represents a current table and has a record.
getType()	This method returns the object type.
doUpdate()	This method updates the database.
doInsert()	This method inserts an object into the database.
doRemove()	This method removes an object from the database.
doDelete()	This method is synonymous with the doRemove method. This method simply has a more intuitive name.
doAction()	This method executes a method defined on the object.
toArray()	This method converts an object to a JavaScript array.
push()	This method adds an item to an array.
pop()	This method removes an item from an array.
join()	This method returns a string of all the entries in the array.
shift()	This method returns a string of all the entries in the array.
unshift()	This method returns a string of all the entries in the array.

setType()	This method sets the object type.
setValue()	This method sets the object to a given value.
doSave()	This method inserts a record if it is new and updates the record if it already exists.
JSDate()	This method returns a JavaScript Date object.

Example

This example does the following:

- Creates a Datum object from the contacts table
- Selects records from this contacts table starting with the letter B
- Displays the contacts matching the select query

This example requires the following sample data:

- A valid query against the contacts table (for example, contact.name # "B")

```
var q = new Datum( "contacts" );
if ( ( q.soSelect( "contact.name # \"B\"" ) ) == true )
{
  print( q.getText() );
}
else
  print( "No contacts start with B" );
```

JavaScript object: Header

The following JavaScript object is unique to HP Service Manager.

With no arguments, the Header() constructor creates an empty Header object.

With the HTTP header arguments, the Header() constructor creates a Header object containing the provided header types and values. You must enclose header types and header values in quotation marks.

Constructor

```
new Header();
new Header(Header type, Header value);
```

Arguments

The following arguments are valid for this object:

Argument	Description
Header type	The HTTP header type such as "Accept-Encoding" or "Content-Type".
Header value	The HTTP header value such as "gzip,deflate" or "text/html".

Properties

None

Methods

None

Example

This example does the following:

- Creates a Header object using the *Header type* and *Header value* arguments
- Creates a second empty Header object
- Defines the properties of the second Header object
- Creates an empty JavaScript array
- Adds the two Header objects to the JavaScript array

This example requires the following sample data:

- A valid header

```
var h = new Header( "Accept-Encoding", "gzip,deflate" );
var hd = new Header();
hd.name = "Content-Type";
hd.value = "text/html";
var headers = new Array();
headers.push( h );
headers.push( hd );
```

JavaScript object: SCDatum

The following JavaScript object is unique to HP Service Manager.

This object is an alias of the SCFile object.

With no arguments, the SCDatum() constructor creates an empty SCDatum object.

With an object argument, the SCDatum() constructor creates an SCDatum object containing the provided record, structure, or array. You must enclose table names and arrays in quotation marks.

This object's methods use the Service Manager-defined global return codes.

Constructor

```
new SCDatum();
new SCDatum(object);
```

Arguments

The following arguments are valid for this object:

Argument	Data type	Description
object	String	The table name, record, structure, or array you want to query or update.

Properties

None

Methods

The following methods are valid for this object:

Defined method	Description
getText()	This method returns a text representation of the object.
getXML()	This method returns an XML representation of the object.
getSize()	This method returns the size of the object.
getMessages()	This method returns an array of messages generated by the Document Engine from a previous SCFile.doAction().
length()	This method returns the length of the object.
getNext()	This method moves to the next record if the object represents a result set.
getPrev()	This method moves to the previous record if the object represents a result set.
getFirst()	This method moves to the first record if the object represents a result set.
getLast()	This method moves to the last record if the object represents a result set.
doSelect()	This method makes a query. It no longer requires that the first argument be the name of the table if the object is constructed with the name of the table.
isRecord()	This method checks if an object represents a current table and has a record.
getType()	This method returns the object type.
doUpdate()	This method updates the database.
doInsert()	This method inserts an object into the database.
doRemove()	This method removes an object from the database.
doDelete()	This method is synonymous with the doRemove method. This method simply has a more intuitive name.
doAction()	This method executes a method defined on the object.
toArray()	This method converts an object to a JavaScript array.
push()	This method adds an item to an array.
pop()	This method removes an item from an array.
join()	This method returns a string of all the entries in the array.
shift()	This method returns a string of all the entries in the array.
unshift()	This method returns a string of all the entries in the array.

setType()	This method sets the object type.
setValue()	This method sets the object to a given value.
doSave()	This method inserts a record if it is new and updates the record if it already exists.
JSDate()	This method returns a JavaScript Date object.

Example

This example does the following:

- Creates an SCDatum object from the contacts table
- Selects records from this contacts table starting with the letter B
- Displays the contacts matching the select query

This example requires the following sample data:

- A valid query against the contacts table (for example, contact.name # "B")

```
var q = new SCDatum( "contacts" );
if ( ( q.doSelect( "contact.name # \"B\"" ) ) == RC_SUCCESS )
{
  print( q.getText() );
}
else
  print( "Return code is: " + q.RCtoString );
```

JavaScript object: SCFile

The following JavaScript object is unique to Service Manager.

With no parameters, the SCFile() constructor creates an empty SCFile object.

With an object parameter, the SCFile() constructor creates an SCFile object containing the provided record, structure, or array. You must enclose table names and arrays in quotation marks.

This object's methods use the Service Manager-defined global return codes.

Constructor

```
new SCFile();
new SCFile(object);
new SCFile( object, SCFILE_READONLY );
```

Arguments

The following arguments are valid for this object:

Argument	Data type	Description

object	String	The table name, record, structure, or array you want to query or update.
SCFILE_ READONLY	Number	Causes the file object to become read-only. This constant is only supported as a second parameter, with the file name specified as the first parameter.

Properties
None

Properties
None

Methods
The following methods are valid for this object:

Defined method	Description
deleteAttachment()	This method deletes a specific attachment associated with the current file record.
deleteAttachments()	This method deletes all of the attachments associated with the current file record.
doAction()	This method executes a method defined on the object.
doCount()	This method returns the number of records for the provided query.
doDelete()	This method is synonymous with the doRemove() method. This method simply has a more intuitive name.
doInsert()	This method inserts an object into the database.
doPurge()	This method purges a set of records directly in the back-end RDBMS.
doRemove()	This method removes an object from the database.
doSave()	This method inserts a record if it is new and updates the record if it already exists.
doSelect()	This method makes a query. It no longer requires that the first argument be the name of the table if the object is constructed with the name of the table.
doUpdate()	This method updates the database.
getAttachment()	This method gets a specific attachment associated with the current file record.
getAttachments()	This method gets all of the attachments associated with the current file record.

getFirst()	This method moves to the first record if the object represents a result set.
getLast()	This method moves to the last record if the object represents a result set.
getLastRC()	This method returns the last return code generated by the SCFile and SCDatum objects.
getMessages()	This method returns an array of messages generated by the Document Engine from a previous doAction().
getNext()	This method moves to the next record if the object represents a result set.
getPrev()	This method moves to the previous record if the object represents a result set.
getPurgedRowCount ()	This method retrieves the number of records deleted from a table by the doPurge().
getSize()	This method returns the size of the object.
getText()	This method returns a text representation of the object.
getType()	This method returns the object type.
getXML()	This method returns an XML representation of the object.
insertAttachment()	This method creates a new attachment associated with the current file record.
isRecord()	This method checks if an object represents a current table and has a record.
join()	This method returns a string of all the entries in the array.
JSDate()	This method returns a JavaScript Date object.
length()	This method returns the length of the object.
pop()	This method removes an item from an array.
push()	This method adds an item to an array.
setFields()	This method causes any subsequent doSelect() calls to fetch only the specified fields.
setRecord()	This method sets the field values from a given XML string.
setOrderBy()	This method causes any subsequent doSelect() call on a SCFile object to fetch only the specified fields by the specified sort sequences of the SCFileObject.
setType()	This method sets the object type.

setValue()	This method sets the object to a given value.
shift()	This method returns a string of all the entries in the array.
toArray()	This method converts an object to a JavaScript array.
unshift()	This method returns a string of all the entries in the array.
updateAttachment()	This method updates a specific attachment associated with the current file record.

Example

This example does the following:

- Creates an SCFile object from the contacts table
- Selects records from this contacts table starting with the letter B
- Displays the contacts matching the select query

This example requires the following sample data:

- A valid query against the contacts table (for example, contact.name # "B")

```
var q = new SCFile("contacts");
if ( ( q.doSelect( "contact.name # \"B\"" ) ) == RC_SUCCESS )
{
  print(q.getText());
}
else
  print( "Return code is: " + q.RCtoString );
```

JavaScript method: SCFile.deleteAttachment(attachID)

This method deletes a specific attachment associated with the current file record.

Values passed to deleteAttachment() will normally be ones obtained from a previous insertAttachment(), updateAttachment(), or getAttachments() operation.

Before calling deleteAttachment, you must establish a current record using one of the positioning methods, such as doSelect(), getNext(), etc.

Syntax

```
SCFile.deleteAttachment( attachID );
```

Arguments

deleteAttachment() takes one argument, which is a string containing an attachment ID of the form "cid:xxxxx".

Return values

If successful, deleteAttachment() returns RC_SUCCESS. Otherwise. it returns an error return code. For example, it returns RC_NO_MORE if the attachment was not found.

Example

This example does the following:

- Selects one record from the `probsummary` file
- Deletes one attachment associated with the current record

This example requires the following sample data:

- Record number `IM10001` in the `probsummary` table

```
function createTestAttachment()
{
  var attachmentObj = new Attachment();

  attachmentObj.type  = "text/plain";
  attachmentObj.name  = "MyAttachment.txt";
  attachmentObj.value = "My text attachment";

  var f = new SCFile( 'probsummary' );

  var rc = f.doSelect( 'number = "IM10001"' );

  var attachmentID = f.insertAttachment( attachmentObj );

  return attachmentID;
}

var attachmentID = createTestAttachment();

var f = new SCFile( 'probsummary' );

var rc = f.doSelect( 'number = "IM10001"' );

var rc = f.deleteAttachment( attachmentID );
```

JavaScript method: SCFile.deleteAttachments()

Deletes all of the attachments associated with the current file record.

Before calling deleteAttachment, you must establish a current record using one of the positioning methods, such as doSelect(), getNext(), etc.

Syntax

```
SCFile.deleteAttachments()
```

Arguments

deleteAttachments() takes no arguments and returns no arguments.

Return values

If successful, deleteAttachments() deletes all of the attachments associated with the current record.

Example

This example does the following:

- Selects one record from the `probsummary` file
- Deletes all attachments associated with the current record
- Displays the number of attachments associated with the current record

This example requires the following sample data:

- Record number `IM10001` in the `probsummary` file

```
var f = new SCFile( 'probsummary' );

var rc = f.doSelect( 'number = "IM10001"' );

f.deleteAttachments();

var attachments = f.getAttachments();

print( attachments.length );  // will be zero
```

JavaScript method: SCFile.doAction()

This method executes a Document Engine action using any new field values defined in a Service Manager file object. You must define a Service Manager file object and new values for required fields.

Syntax

`SCFile.doAction( docEngineAction );`

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
docEngineAction	String	This argument specifies the Document Engine action you want the script to execute on the object. This argument must contain a valid Service Manager Document Engine action appropriate for the Service Manager object.

Return values

RC_SUCCESS or one of the other global return code values.
The method returns RC_SUCCESS if the method successfully runs the Document Engine action or returns one of the error global return code values if the method cannot run the Document Engine action.

Example

This example does the following:

- Creates a new incident record from variable values
- Resolves the incident record

This example requires the following sample data:

- The "Use Resolved Status?" option enabled in the Incident Management Environment record
- A valid value for the number field (for example, "IM11112")

```
var numberValue;
var callbackContact;
var categoryValue;
var assignmentValue;
var descriptionValue;
var actionType;
var resultionCodeValue;
var resolutionValue;

function insertIncident( num, name, cat, group, desc )
{
 print( "Creating new incident record..." );
 var newIncident = new SCFile( "probsummary" );
 newIncident.number = num;
 newIncident.callback_contact = name;
 newIncident.category = cat;
 newIncident.assignment = group;
 newIncident.description = desc;
 var rc = newIncident.doInsert();
 if ( rc == RC_SUCCESS )
 {
  print( "Success. Created new record " + newIncident.getText() );
  return newIncident.number
 }
 else
 {
  print( "Could not create record. " + RCtoString( rc ) );
  return null
 }
}

function resolveIncident( id, action, rescode, res )
{
 print( "Resolving incident " + id + "..." );
 var findIncident = new SCFile ( "probsummary" );
 var f = findIncident.doSelect( "number=\"" + id + "\"" );
 if ( f == RC_SUCCESS )
 {
  print( "Success. Found interaction record..." );
  findIncident.resolution_code = rescode;
  findIncident.resolution = res;
  var a = findIncident.doAction( action );
  print( "The resolve Incident return value is " + RCtoString( a )
);
  print( "The incident record is " + findIncident.getText() )
  print( "The value of the status field is: " +
findIncident.problem_status );
```

```
    return findIncident
   }
   else
   {
    print( "Could not find record. " + RCtoString( f ) );
    return null
   }
  }

  numberValue = "IM11121"
  callbackContact = "ADMINISTRATOR, SYSTEM";
  categoryValue = "network";
  assignmentValue = "HELPDESK";
  descriptionValue = null;
  interactionID = insertIncident( numberValue, callbackContact,
  categoryValue, assignmentValue, descriptionValue );
  actionType = "resolve";
  resultionCodeValue = "Advice & Guidance";
  resolutionValue = null;
  resolveIncident( interactionID, actionType, resultionCodeValue,
  resolutionValue );
```

JavaScript method: SCFile.doCount()

This method returns the number of records specified by the query.

Note: You should not use this method on an SCFile object for which you have performed a previous doSelect() if you intend to iterate over the results of the doSelect().

Syntax

```
SCFile.doCount( "query" );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
query	String	This argument specifies the query you want to use to search for Service Manager records. You must use the slash character to escape out quotation marks and any special characters restricted from JavaScript.

Return values

A number representing the count of records.

Example

This example does the following:

- Determines the number of contact records.
- Displays the contact record as a text string.

This example requires the following sample data:

- A valid contact name (for example, "ADMINISTRATOR, SYSTEM").
- An invalid contact name (for example, "NOT A, CONTACT").

```
var contactName;

function countContacts( location )
{
    print( "Counting contacts at location: " + location + "..." );
    var contactList = new SCFile( "contacts" );
    var nCount = contactList.doCount( "location=\""+ location +
                 "\"" );
    var rc = getLastRC();
    if ( rc == RC_SUCCESS )
    {
      return nCount;
    }
    else
    {
      print( "Could not count records. " + RCtoString( rc ) );
      return -1;
    }
}
```

JavaScript method: SCFile.doDelete()

This method removes a HP Service Manager record with any new field values defined in a Service Manager file object. You must define a Service Manager file object and new values for fields. This method is a synonym of the doRemove() method, and related to the more extensive doPurge() method.

Syntax

```
SCFile.doDelete();
```

Arguments

There are no arguments for this method.

Return values

RC_SUCCESS or one of the other global return code values.
The method returns RC_SUCCESS if the method successfully removes a record or returns one of the error global return code values if the method cannot remove the record.

Example

This example does the following:

- Searches the probsummary table for any interaction record you define in the search variable
- Removes the interaction record

This example requires the following sample data:

- A valid value for the number field (for example, "IM11111")

```
var numberValue;

function removeInteraction( num )
{
 print( "Removing interaction record..." );
 var interactionRecord = new SCFile( "probsummary" );
 var findRecord = interactionRecord.doSelect( "number=\"" + num +
"\"");
 print( "The interaction record is " + interactionRecord );
 var rc = interactionRecord.doDelete();
 if ( rc == RC_SUCCESS )
 {
  print( "Success. Removed record " + interactionRecord.getText() );
  return interactionRecord
 }
 else
 {
  print( "Could not remove record. " + RCtoString( rc ) );
  return null
 }
}

numberValue = "IM11111"
removeInteraction( numberValue );
```

JavaScript method: SCFile.doInsert()

This method creates a Service Manager record with any new field values defined in a file object. You must define a Service Manager file object and new values for the fields.

Syntax

```
SCFile.doInsert();
```

Arguments

There are no arguments for this method.

Return values

RC_SUCCESS or one of the other global return code values.
The method returns RC_SUCCESS if the method successfully creates a record or returns one of the error global return code values if the method cannot create the record.

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Displays the new incident record as a text string

This example requires the following sample data:

- A valid value for the number field (for example, "IM11111")
- A valid value for the callback.contact field (for example, "ADMINISTRATOR, SYSTEM")
- A valid value for the category field (for example, "network")

```
var numberValue;
var callbackContact;
var categoryValue;
var descriptionValue;

function insertContact( num, name, cat )
{
 print( "Creating new interaction record..." );
 var newInteraction = new SCFile( "probsummary" );
 newInteraction.number = num;
 newInteraction.callback_contact = name;
 newInteraction.category = cat;
 var rc = newInteraction.doInsert();
 if ( rc == RC_SUCCESS )
 {
  print( "Success. Created new record " + newInteraction.getText()
);
  return newInteraction
 }
 else
 {
  print( "Could not create record. " + RCtoString( rc ) );
  return null
 }
}

numberValue = "IM11112"
callbackContact = "ADMINISTRATOR, SYSTEM";
categoryValue = "network";
insertContact( numberValue, callbackContact, categoryValue,
descriptionValue );
```

JavaScript method: SCFile.doPurge()

This method enables you to purge a set of HP Service Manager records directly in the back-end RDBMS while bypassing record level processing. The doPurge() method is an alternative to doDelete() or doRemove(). It enables the back-end RDBMS to do most of the processing, thereby improving execution times when deleting a large amount of data from a single table.

The doDelete() method requires you to select a list of records from the database, iterate over each record in the list and perform a “delete” on a record by record basis. When the delete is performed, Service Manager respects workflow actions as defined in record level triggers. The doPurge() method reduces the overhead of much of this by sending one SQL statement to the RDBMS to purge the data. As a result of the RDBMS processing the record set purge in one transaction,

Service Manager does not have control to process record level triggers on the data being purged. Standard Service Manager trigger processing will not occur on these records.

You should use doPurge() only when you need to quickly delete a large record set from a single Service Manager table, and it does not matter whether Service Manager triggers are processed.

Warning: Purging data has some risks. Use doPurge() with caution. It is the equivalent of issuing a SQL statement to the RDBMS such as, `DELETE FROM "SM_TABLE_NAME" WHERE "COLOUMN_NAME" = 'value'`.

Some issues to understand and consider before using this method:

- Data referential integrity could be an issue by purging data that is related to other types of Service Manager data. This method does not execute normal record level workflow processing in the applications layer. When you turn off triggers and run doPurge(), you will bypass normal workflow processing in addition to record level triggers.
- No Service Manager rollback or undo is available after you run doPurge().
- No Service Manager audit or other tracking is available to indicate changes made by doPurge(). Consider coding your own audit mechanism.

You can retrieve the number of records deleted from a table by doPurge() by using the JavaScript method: getPurgedRowCount()

Note: When you use doPurge(), Service Manger deletes a set of records. This causes a delete to the back-end database. When Service Manager attempts to delete a large set of records, the database could run out of space for this activity and cause an error. Each supported database platform provides some type of transaction or undo log to back up data changes and allow a rollback in case of errors. If you encounter such an error, please contact your database administrator and ask them to increase the size available for this purpose.

Some examples of the errors you could encounter are:

Oracle:SQL code=30036 message=ORA-30036: unable to extend segment by % in undo tablespace

SQL Server SQL State: 42000-9002 Message: [Microsoft][SQL Native Client][SQL Server]The transaction log for database '%' is full

DB2 SQL State: 57011--964 Message: [IBM][CLI Driver][DB2/NT] SQL0964C The transaction log for the database is full. SQLSTATE=57011

Syntax

```
SCFile.doPurge( "query" );
```

Arguments

The following arguments are valid for this method:

Argument	Description
query	String, Boolean, or QueryCond object This argument specifies the query you want to use to search for Service Manager records. You must use the slash character to escape out quotation marks and any special characters restricted from JavaScript. Boolean values (true or false) or a QueryCond object are also acceptable.

Return values

Returns RC_SUCCESS if successful and RC_ERROR on failure.

The following conditions will cause a doPurge() failure:

- Calling doPurge() on an uninitialized SCFile or a read-only SCFile
- Calling doPurge() with an empty query argument
- Calling doPurge() with an invalid argument type (not a string)
- Calling doPurge() with one (or more) invalid (or unmapped) fields.
  A basic validation is performed on the field names to ensure they exist in the DBDICT. This validation will not cause a failure of the method; it will return RC_SUCCESS. However, in this case, it will default to a “false” query and no records will be deleted.
- Calling doPurge() with an unsupported complex type SCFile relation.
  See the limitations below for supported types.
- Calling doPurge() with an SCFile object which has “delete” triggers defined and enabled.

When the method fails, it has no effect.

Limitations

Only simple SCFile relations are supported:

- Must not be a join file or a merge file
- Must have only 1 main table alias (m1)
- Must not have any alias tables (a1,a2,….)
- No LDAP mapped fields
- The table must not have “delete” triggers defined and enabled

Example

This example does the following:

- Searches the kmknowledgebaseupdates table for records where docid="docid1"
- Deletes whatever it finds.

This example requires the following sample data:

- Records in kmknowledgebaseupdates table where docid = "docid1"

```
var fileName = "kmknowledgebaseupdates";
var recordList = new SCFile(fileName);
var deleteRC = recordList.doPurge("docid = \"docid1\"");
```

JavaScript method: SCFile.doRemove()

This method removes a single HP Service Manager record matching the field values defined in the Service Manager file object. To remove multiple records use the doSelect method to store a record list as a JavaScript object, and then use the getNext, getPrev, getFirst, or GetLast methods to bring a particular record into memory for removal. It requires defining a Service Manager file object. This

method is a synonym of the doDelete() method, and related to the more extensive doPurge() method.

Syntax

```
SCFile.doRemove();
```

Arguments

There are no arguments for this method.

Return values

RC_SUCCESS or one of the other global return code values.
The method returns RC_SUCCESS if the method successfully removes a record or returns one of the error global return code values if the method cannot remove the record.

Example

This example does the following:

- Searches the probsummary table for any interaction record you define in the search variable
- Removes the interaction record

This example requires the following sample data:

- A valid value for the number field (for example, "IM11111")

```
var numberValue;

function removeInteraction( num )
{
 print( "Removing interaction record..." );
 var interactionRecord = new SCFile( "probsummary" );
 var findRecord = interactionRecord.doSelect( "number=\"" + num +
"\"");
 print( "The interaction record is " + interactionRecord );
 var rc = interactionRecord.doRemove();
 if ( rc == RC_SUCCESS )
 {
  print( "Success. Removed record " + interactionRecord.getText() );
  return interactionRecord
 }
 else
 {
  print( "Could not remove record. " + RCtoString( rc ) );
  return null
 }
}

numberValue = "IM11111"
removeInteraction( numberValue );
```

JavaScript method: SCFile.doSave()

This method adds to or updates the Service Manager database with any new field values defined in a Service Manager file object. It requires defining a Service Manager file object and defining new values for fields.

Syntax

```
SCFile.doSave();
```

Arguments

There are no arguments for this method.

Return values

RC_SUCCESS or one of the other global return code values.
The method returns RC_SUCCESS if the method successfully updates a record or returns one of the error global return code values if the method cannot update the record.

Example

This example does the following:

- Searches the contacts table for any contact name you define in the search variable
- Displays the contact record as a text string
- Updates the value of the user.type field
- Saves the Service Manager record

This example requires the following sample data:

- A valid contact name (for example, "ADMINISTRATOR, SYSTEM")
- A valid field value update (for example, set the "user.type" field to "site")

```
var contactName;

function updateContact( name )
{
 print( "Searching for contact: " + name + "..." );
 var contactList = new SCFile( "contacts" );
 var isContact = contactList.doSelect( "contact.name=\""+ name +
"\"" );
 if ( isContact == RC_SUCCESS )
 {
  print( "Success. found " + name + " in contact record:\n" +
contactList.getText() );
  print( "The current contents of the user.type field are: " +
contactList.user_type );
  print( "Updating the user.type field..." );
  contactList.user_type = "site";
  print( "Saving record..." );
  var rc = contactList.doSave();
  print( "The return code value for the doSave() method is: " +
RCtoString( rc ) );
```

```
  print( "The contents of the user.type field are now: " +
contactList.user_type );
  return contactList
 }
 else
 {
  print( "Could not find contact. " + RCtoString( isContact ) );
  return null
 }
}

contactName = "ADMINISTRATOR, SYSTEM";
updateContact( contactName );
```

JavaScript method: SCFile.doSelect()

This method queries for records in an HP Service Manager object. The SCFile object has the following format: `table={["field value",{"array value","array value"},...]}`

For performance improvement, you can use SCFile.setFields. method to limit the fields that doSelect returns from the database.

Important Note: This method is used for retrieving normal data, not for retrieving attachments. If you use this method to retrieve attachments, you could corrupt the SYSATTACHMENTS table. If you need to retrieve attachments, use one of the following functions to get the attachments associated with the current file record: `SCFile.getAttachment(attachID)` or `SCFile.getAtttachments()`.

Syntax

```
SCFile.doSelect( "query" );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
query	String	This argument specifies the query you want to use to search for Service Manager records. You must use the slash character to escape out quotation marks and any special characters restricted from JavaScript.

Return values

An SCFile object containing records and RC_SUCCESS or one of the other global return code values.

The method returns an SCFile object containing the records matching the query and a global return code value of RC_SUCCESS or returns one of the error global return code values if the method cannot return any records.

Example

This example does the following:

- Searches the contacts table for any contact name you define in the search variable
- Displays the contact record as a text string

This example requires the following sample data:

- A valid contact name (for example, "ADMINISTRATOR, SYSTEM")
- A invalid contact name (for example, "NOT A, CONTACT")

```
var contactName;

function findContactName( name )
{
 print( "Searching for contact: " + name + "..." );
 var contactList = new SCFile( "contacts" );
 var findContact = contactList.doSelect( "contact.name=\""+ name +
"\"" )
 if ( findContact == RC_SUCCESS )
 {
  print( "Success. found " + name + " in contact record:\n" +
contactList.getText() );
  return contactList
 }
 else
 {
  print( "Could not find contact. " + RCtoString( findContact ) );
  return null
 }
}

contactName = "ADMINISTRATOR, SYSTEM";
findContactName( contactName );

contactName = "NOT A, CONTACT";
findContactName( contactName );
```

JavaScript method: SCFile.doUpdate()

This method updates a Service Manager record with any new field values defined in a Service Manager file object. It requires defining a Service Manager file object and defining new values for fields.

Syntax

```
SCFile.doUpdate();
```

Arguments

There are no arguments for this method.

Return values

RC_SUCCESS or one of the other global return code values.

The method returns RC_SUCCESS if the method successfully updates a record or returns one of the error global return code values if the method cannot update the record.

Example

This example does the following:

- Searches the contacts table for any contact name you define in the search variable
- Displays the contact record as a text string

This example requires the following sample data:

- A valid contact name (for example, "ADMINISTRATOR, SYSTEM")
- A valid field value update (for example, set the "user.type" field to "site")

```
var contactName;

function updateContact( name )
{
 print( "Searching for contact: " + name + "..." );
 var contactList = new SCFile( "contacts" );
 var isContact = contactList.doSelect( "contact.name=\""+ name +
"\"" );
 if ( isContact == RC_SUCCESS )
 {
  print( "Success. found " + name + " in contact record:\n" +
contactList.getText() );
  print( "The current contents of the user.type field are " +
contactList.user_type );
  print( "Updating the user.type field..." );
  contactList.user_type = "site";
  contactList.doUpdate();
  print( "The contents of the user.type field are now " +
contactList.user_type );
  return contactList
 }
 else
 {
  print( "Could not find contact. " + RCtoString( isContact ) );
  return null
 }
}

contactName = "ADMINISTRATOR, SYSTEM";
updateContact( contactName );
```

JavaScript method: SCFile.getAttachment(attachID)

Gets a specific attachment associated with the current file record.

Values passed to getAttachment() will normally be ones obtained from a previous insertAttachment (), updateAttachment(), or getAttachments() operation.

Before calling getAttachment, you must establish a current record using one of the positioning methods, such as doSelect(), getNext(), etc.

Syntax

```
SCFile.getAttachment( attachID );
```

Arguments

getAttachment() takes one argument, which is a string containing an attachment ID of the form "cid:xxxxx".

Return values

If successful, getAttachment() returns an Attachment object. Otherwise, it returns null.

Example

This example does the following:

- Selects one record from the `probsummary` table.
- Gets a specific attachment associated with the current record

This example requires the following sample data:

- Record number `IM10001` in the `probsummary` table

```
var f = new SCFile( 'probsummary' );

var rc = f.doSelect( 'number = "IM10001"' );

var attachmentObj = f.getAttachment( attachmentID );
```

JavaScript method: SCFile.getAttachments()

This method gets all of the attachments associated with the current file record.

Before calling getAttachments, you must establish a current record using one of the positioning methods, such as doSelect(), getNext(), etc.

Syntax

```
SCFile.getAttachments()
```

Arguments

The getAttachments method takes no arguments.

Return values

If successful, getAttachments() returns an array of attachment objects. The array will be empty if there are no attachments for the record. The attachment objects retrieve here will only contain metadata. The Attachment.value will be NULL. To get each attachment's binary data, you will need to use SCFile.getAttachment(attachID) method where attachID contains the href property of the attachment object.

Example

This example does the following:

- Selects one record from the probsummary table
- Gets all of the attachments associated with the current record

This example requires the following sample data:

- Record number `IM10001` in the `probsummary` table

```
var f = new SCFile( 'probsummary' );

var rc = f.doSelect( 'number = "IM10001"' );

var attachments = f.getAttachments();

     print( attachments.length );
     for ( var attachment in attachments )
     {
         print( attachments[ attachment ].name );
     }
```

JavaScript method: SCFile.getFirst()

This method returns the first record in a record list stored in an SCFile object.You must use one of the other SCFile methods to create a record list.

Syntax

```
SCFile.getFirst();
```

Arguments

There are no arguments for this method.

Return values

An SCFile object containing the first record and RC_SUCCESS or one of the other global return code values.

The method returns an SCFile object containing the first record in a record list and a global return code value of RC_SUCCESS or returns one of the error global return code values if the method cannot find the first record.

Example

This example does the following:

- Queries the contacts table for any contact name you define in the search variable
- Displays the last record in the record list
- Displays the previous record in the record list
- Displays the first contact in the record list

This example requires the following sample data:

- A contact query that produces a record list (for example, "B")
- A contact query that produces a single record (for example, "F")

```
var contactQuery;

function findFirstContact( query )
{
 print( "Searching for contacts starting with " + query + "..." );
 var contactFile = new SCFile( "contacts" );
 var findContact = contactFile.doSelect( "contact.name#\""+ query +
"\"" );
 if ( findContact == RC_SUCCESS )
 {
  print( "Success. Found contacts starting with " + query + "." );
  var findLastRecord = contactFile.getLast();
  print( "The last record is:\n" + contactFile );
  var findPrevRecord = contactFile.getPrev();
  print( "The previous record is:\n" + contactFile );
  var findFirstRecord = contactFile.getFirst();
  print( "The first record is:\n" + contactFile );
 }
 else
 {
  print( "Could not find contacts starting with " + query + ". " +
RCtoString( findContact ) );
  return null
 }
}
contactQuery = "H";
findFirstContact( contactQuery );
```

JavaScript method: SCFile.getLast()

This method returns the last record in a record list stored in an SCFile object. You must use the other SCFile methods to create a record list.

Syntax

```
SCFile.getLast();
```

Arguments

There are no arguments for this method.

Return values

An SCFile object containing the last record and RC_SUCCESS or one of the other global return code values.

The method returns an SCFile object containing the last record in a record list and a global return code value of RC_SUCCESS or returns one of the error global return code values if the method cannot find the last record.

Example

This example does the following:

- Queries the contacts table for any contact name you define in the search variable
- Displays the first contact in the record list
- Displays the last record in the record list

This example requires the following sample data:

- A contact query that produces a record list (for example, "B")
- A contact query that produces a single record (for example, "F")

```
var contactQuery;

function findLastContact( query )
{
 print( "Searching for contacts starting with " + query + "..." );
 var contactFile = new SCFile( "contacts" );
 var findContact = contactFile.doSelect( "contact.name#\""+ query +
"\"" );
 if ( findContact == RC_SUCCESS )
 {
  print( "Success. Found contacts starting with " + query + ". The
first record is:\n" + contactFile );
  var findLastRecord = contactFile.getLast();
  print( "The last record is:\n" + contactFile );
  return contactFile
 }
 else
 {
  print( "Could not find contacts starting with " + query + ". " +
RCtoString( findContact ) );
  return null
 }
}
contactQuery = "H";
findLastContact( contactQuery );

contactQuery = "F";
findLastContact( contactQuery );
```

JavaScript method: SCFile.getLastRC()

This method returns the most recent global return code value generated by the SCFile object. You can then use the RCtoString global method to convert the return code into a string.

Syntax

```
SCFile.getLastRC();
```

Arguments

There are no arguments for this method.

Return values

One of the global return code values.
The method returns the most recent global return code value generated by the SCFile object.

Example

This example does the following:

- Inserts a new record into a local variable
- Displays the return code of the insert operation

This example requires the following sample data:

- A valid record in a local variable (for example, "$L.file")

```
system.vars.$L_file.doInsert();
var rc = getLastRC();
if ( rc == RC_SUCCESS )
 {
  print( "Insert of document succeeded" );
 }
 else if ( getLastRC() == false )
 print( "Error " + RCtoString(rc) + " occurred trying to insert the
record" );
```

JavaScript method: SCFile.getMessages()

Returns an array of messages generated by the Document Engine from a previous SCFile.doAction ().

Syntax

```
SCFile.getMessages();
```

Arguments

There are no arguments allowed for this method.

Return values

Returns an array of messages from previous SCFile.doAction() call.

Example

This example does the following:

- Uses the Document Engine add to create a new incident record from variable values.
- Shows a validation failed due to invalid assignment group.
- SCFile.doAction() returns 71, "validation failure".
- SCFile.getMessages() returns messages generated from the Document Engine.

```
var numberValue;
var callbackContact;
var categoryValue;
```

```
var assignmentValue;
var descriptionValue;
var actionType;
var resultionCodeValue;
var resolutionValue;

function insertIncident( num, name, cat, group, desc )
{
 print( "Creating new incident record..." );
 var newIncident = new SCFile( "probsummary" );
 newIncident.number = num;
 newIncident.callback_contact = name;
 newIncident.category = cat;
 newIncident.assignment = group;
 newIncident.description = desc;
 var rc = newIncident.doAction('add');
 if ( rc == RC_SUCCESS )
 {
  print( "Success. Created new record " + newIncident.getText() );
  return newIncident.number
 }
 else
 {
  print( "Could not create record. " + RCtoString( rc ) );
  print( "Reason: "+ newIncident.getMessages() );
  return null
 }
}

numberValue = "IM11121"
callbackContact = "ADMINISTRATOR, SYSTEM";
categoryValue = "network";
assignmentValue = "HELPDESK";
descriptionValue = null;
interactionID = insertIncident( numberValue, callbackContact,
categoryValue, assignmentValue, descriptionValue );
```

Output

```
        Category specified is invalid.   Reason: Category specified is
invalid. Could not create record. Validation failed   creating new
incident record...
```

JavaScript method: SCFile.getNext()

This method returns the next record in a record list stored in an SCFile object. It requires the use of the other SCFile methods to create a record list.

Syntax

```
SCFile.getNext();
```

Arguments

There are no arguments for this method.

Return values

An SCFile object containing the next record and RC_SUCCESS or one of the other global return code values.

The method returns an SCFile object containing the next record in a record list and a global return code value of RC_SUCCESS or returns one of the error global return code values if the method cannot find the next record.

Example

This example does the following:

- Queries the contacts table for any contact name you define in the search variable
- Displays the first contact in the record list
- Displays the next record in the record list

This example requires the following sample data:

- A contact query that produces a record list (for example, "B")
- A contact query that produces a single record (for example, "F")

```
var contactQuery;

function findNextContact( query )
{
 print( "Searching for contacts starting with " + query + "..." );
 var contactFile = new SCFile( "contacts" );
 var findContact = contactFile.doSelect( "contact.name#\""+ query +
"\"" );
 if ( findContact == RC_SUCCESS )
 {
  print( "Success. Found contacts starting with " + query + ". The
first record is:\n" + contactFile.getText() );
  var findNextRecord = contactFile.getNext();
  print( "The next record is:\n" + contactFile.getText() );
  return contactFile
 }
 else
 {
  print( "Could not find contact " + query + ". " + RCtoString(
findContact ) );
  return null
 }
}
contactQuery = "B";
findNextContact( contactQuery );

contactQuery = "F";
findNextContact( contactQuery );
```

JavaScript method: SCFile.getPrev()

This method returns the previous record in a record list stored in an SCFile object. You must use of the other SCFile methods to create a record list.

Syntax

```
SCFile.getPrev();
```

Arguments

There are no arguments for this method.

Return values

An SCFile object containing the previous record and RC_SUCCESS or one of the other global return code values.

The method returns an SCFile object containing the previous record in a record list and a global return code value of RC_SUCCESS or returns one of the error global return code values if the method cannot find the previous record.

Example

This example does the following:

- Queries the contacts table for any contact name you define in the search variable
- Displays the first contact in the record list
- Displays the last record in the record list
- Displays the previous record in the record list

This example requires the following sample data:

- A contact query that produces a record list (for example, "H")

```
var contactQuery;

function findPrevContact( query )
{
 print( "Searching for contacts starting with " + query + "..." );
 var contactFile = new SCFile( "contacts" );
 var findContact = contactFile.doSelect( "contact.name#\""+ query +
"\"" );
 if ( findContact == RC_SUCCESS )
 {
  print( "Success. Found contacts starting with " + query + ". The
first record is:\n" + contactFile );
  var findLastRecord = contactFile.getLast();
  print( "The last record is:\n" + contactFile );
  var findPrevRecord = contactFile.getPrev();
  print( "The previous record is:\n" + contactFile );
 }
 else
 {
  print( "Could not find contacts starting with " + query + ". " +
RCtoString( findContact ) );
```

```
    return null
   }
  }
  contactQuery = "H";
  findPrevContact( contactQuery );
```

JavaScript method: SCFile.getPurgedRowCount()

This method enables you to retrieve the number of HP Service Manager records that were deleted from a table after invoking the doPurge() method.

Syntax

```
SCFile.getPurgedRowCount();
```

Arguments

There are no arguments for this method.

Return values

- If successful, getPurgedRowCount() returns the number of records deleted by the last successful doPurge() call (which returned RC_SUCCESS). This value can be 0 if no records were deleted.
- If getPurgedRowCount() is not able to retrieve the record count it will return -1.

The value returned by getPurgedRowCount is unaffected by any SCFile method call other than doPurge. It is also unaffected by a failed call to doPurge (returned RC_ERROR).

Example

This example does the following:

- Searches the kmknowledgebaseupdates table for records where docid="docid1"
- Deletes whatever it finds.

This example requires the following sample data:

- Records in kmknowledgebaseupdates table where docid = "docid1"

```
var fileName = "kmknowledgebaseupdates";
var recordList = new SCFile(fileName);
var deleteRC = recordList.doPurge("docid = \"docid1\"");
if ( deleteRC == RC_SUCCESS )
{
  print ("doPurge succeeded");
  var purgedCount = recordList.getPurgedRowCount();
  if ( purgedCount >= 0 )
  {
    print(purgedCount + " records have been purged");
  }
  else
  {
```

```
      print("Could not retrieve purged record count");
    }
  }
  else
  {
    print ("doPurge failed");
  }
```

JavaScript method: SCFile.getSize()

This method returns the number of array elements in an array stored in a Service Manager Datum object. You can use this method to return the length of Service Manager array fields, which are always stored as Datum objects. To query a particular array field in an SCFile object, you can use the following format: `SCFile_object.array_field_name.getSize()`

Replace any periods in the name of *array_field_name* with underscores. For example, you must convert the Service Manager array field *update.action* to *update_action* when querying it from JavaScript.

Syntax

```
SCFile.Datum_object.getSize();
```

Arguments

There are no arguments for this method.

Return values

A number and RC_SUCCESS or one of the other global return code values.
The method returns the number of array elements in an array stored in a Service Manager Datum object and a global return code value of RC_SUCCESS, or returns one of the error global return code values if the method cannot return the number of array elements.

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Attempts to get the size of the returned SCFile object (the result is always zero)
- Displays the contents of the action field (the action field contains Service Manager array data)
- Displays the size of the action field using the Service Manager getSize() method
- Converts the SCFile object into a JavaScript array using the Service Manager toArray() method
- Displays the size of the new JavaScript array using the core JavaScript length property

This example requires the following sample data:

- A valid incident record number (for example, "IM1005")

```
var incidentQuery;
```

```
function findIncidents( query )
{
 print( "Searching for incident records starting with " + query +
"..." );
 var incidentFile = new SCFile( "probsummary" );
 var findIncident = incidentFile.doSelect( "number#\""+ query + "\""
);
 if ( findIncident == RC_SUCCESS )
 {
  print( "Success. Found incident records starting with " + query +
". The first record is:\n" + incidentFile );
  var objectSize = incidentFile.getSize();
  print( "Running the getSize() method on the returned SCFile object
produces the following result: " + objectSize );
  var actionText = incidentFile.action;
  print( "The contents of the action field is:\n" + actionText );
  var actionSize = incidentFile.action.getSize();
  print( "The size of the action field is: " + actionSize );
  print( "Converting the SCFile object to a JavaScript array..." );
  var incidentArray = incidentFile.toArray();
  print( "The new JavaScript array is:\n" + incidentArray );
  var arraySize = incidentArray.length;
  print( "The size of the converted array is: " + arraySize );
       return incidentFile;
 }
 else
 {
  print( "Could not find incident records starting with " + query +
". " + RCtoString( findIncident ) );
  return null
 }
}

incidentQuery = "IM1005";
findIncidents( incidentQuery );
```

JavaScript method: SCFile.getText()

This method returns a text string representation of the current SCFile object. The text string has the following format:

```
table={["field value",{"array value","array value"},...]}
```

Syntax

```
SCFile.getText();
```

Arguments

There are no arguments for this method.

Return values

A string and RC_SUCCESS or one of the other global return code values.
The method returns a text string containing the current SCFile object and a global return code value of RC_SUCCESS, or returns one of the error global return code values if the method cannot return a string.

Example

This example does the following:

- Searches the contacts table for any contact name you define in the search variable
- Displays the contact record as a text string

This example requires the following sample data:

- A valid contact name (for example, "ADMINISTRATOR, SYSTEM")
- A invalid contact name (for example, "NOT A, CONTACT")

```
var contactName;

function findContactName( name )
{
 print( "Searching for contact: " + name + "..." );
 var contactList = new SCFile( "contacts" );
 var findContact = contactList.doSelect( "contact.name=\""+ name +
"\"" )
 if ( findContact == RC_SUCCESS )
 {
  print( "Success. found " + name + " in contact record:\n" +
contactList.getText() );
  return contactList
 }
 else
 {
  print( "Could not find contact. " + RCtoString( findContact ) );
  return null
 }
}

contactName = "ADMINISTRATOR, SYSTEM";
findContactName( contactName );

contactName = "NOT A, CONTACT";
findContactName( contactName );
```

JavaScript method: SCFile.getType()

This method returns the data type of HP Service Manager datum objects, such as arrays and structured arrays. This method fails on any scalar fields within file records, such as number fields, date fields, string fields, and boolean fields. To determine the type of scalar fields you can use, see the core JavaScript .type property in the *JavaScript Guide* (http://developer.mozilla.org/).

Syntax

```
SCFile.getType();
```

Arguments

There are no arguments for this method.

Return values

A data type and RC_SUCCESS or one of the other global return code values.
The method returns a data type and a global return code value of RC_SUCCESS or returns one of the error global return code values, if the method cannot determine the data type.

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Displays the contents of the incident record object
- Returns the data type of the incident record object using the getType() method
- Displays the contents of the action field object
- Returns the data type of the action field object using the getType() method
- Displays the contents of the category field
- Returns the data type of the category field object using the core JavaScript type property

This example requires the following sample data:

- A valid incident record (for example, "IM1001")

```
var incidentQuery;

function findTypes( query )
{
 print( "Searching for incident records starting with " + query +
"..." );
 var incidentFile = new SCFile( "probsummary" );
 var findIncident = incidentFile.doSelect( "number#\""+ query + "\""
);
 if ( findIncident == RC_SUCCESS )
 {
  print( "Success. Found incident records starting with " + query +
". The first record is:\n" + incidentFile );
  var objectType = incidentFile.getType();
  print( "The object incidentFile is of type: " + objectType );
  var arrayField = incidentFile.action;
  print( "The contents of the action field are:\n" + arrayField );
  fieldType = arrayField.getType();
  print( "The type of the action field is: " + fieldType );
  var textField = incidentFile.category;
  print( "The contents of the category field are:\n" + textField );
  fieldType = textField.getType;
  print( "The type of the category field is: " + fieldType );
 }
 else
```

```
 {
  print( "Could not find incident records starting with " + query +
". " + RCtoString( findIncident ) );
  return null
 }
}

incidentQuery = "IM1001";
findTypes( incidentQuery );
```

JavaScript method: SCFile.getXML()

This method returns an XML object representation of the current SCFile object. The XML object has the following format, which is compatible with the Service Manager Web services API:

```
<model name="table" query="query"><keys><fieldname>key
value</fieldname>...</keys><instance><fieldname>instance
value</fieldname>...</instance></model>
```

Syntax

```
SCFile.getXML();
```

Arguments

There are no arguments for this method.

Return values

An XML object and RC_SUCCESS or one of the other global return code values.
The method returns an XML object containing the current SCFile object and a global return code value of RC_SUCCESS or returns one of the failure global return code values if the method cannot return an XML object.

Example

This example does the following:

- Searches the contacts table for any contact name you define in the search variable
- Displays the contact record as an XML object

This example requires the following sample data:

- A valid contact name (for example, "ADMINISTRATOR, SYSTEM")
- A invalid contact name (for example, "NOT A, CONTACT")

```
var contactName;

function findContact( name )
{
 print( "Searching for contact: " + name + "..." );
 var contactList = new SCFile( "contacts" );
 var isContact = contactList.doSelect( "contact.name=\""+ name +
"\"" )
 if ( isContact == RC_SUCCESS )
```

```
 {
  print( "Success. found " + name + " in contact record:\n" +
contactList.getXML() );
  return contactList
 }
 else
 {
  print( "Could not find contact. " + RCtoString( isContact ) );
  return null
 }
}

contactName = "ADMINISTRATOR, SYSTEM";
findContact( contactName );

contactName = "NOT A, CONTACT";
findContact( contactName );
```

JavaScript method: SCFile.insertAttachment(attachObj)

This method creates a new attachment associated with the current file record. Before calling insertAttachment, you must establish a current record using one of the positioning methods, such as doSelect(), getNext(), etc.

Syntax

```
SCFile.insertAttachment( attachObj );
```

Arguments

The insertAttachment method takes one argument, which is an asttachment object.
The Attachment object must be properly populated prior to calling insertAttachment.
Attachment objects can be obtained via Web Services calls or constructed by JavaScript code.

Required properties

Attachment.name — This is a string containing the name of the attachment, for example, "foo.txt" or "foo.pdf"
Attachment.type — This is a string containing the MIME type of the attachment, such as "text/plain" or "application/pdf"
Attachment.value — This is a binary string containing the actual attachment data. One way to obtain such data is by using the readFile() global method.
The "href" property of the attachment object should not be set for an insert operation. Any value supplied will be ignored. Attachment ID values are assigned by the server when attachments are inserted or updated.

Return values

If successful, insertAttachment() returns a string of the form cid:xxxxxx containing the attachment ID for the inserted attachment.

Example

This example does the following:

- Creates a new attachment object
- Selects one record from the `probsummary` table
- Inserts the attachment object to the current record

This example requires the following sample data:

- Record number `IM10001` in the `probsummary` table

```
var attachmentObj = new Attachment();

attachmentObj.type   = "text/plain";
attachmentObj.name  = "MyAttachment.txt";
attachmentObj.value = "This is a test text attachment";

var f = new SCFile( 'probsummary' );

var rc = f.doSelect( 'number = "IM10001"' );

var attachmentID = f.insertAttachment( attachmentObj );
```

JavaScript method: SCFile.isRecord()

This is a deprecated method as of version 6.1 because it does not use the Service Manager-defined global return codes. It requires using the deprecated File() object with a query or query condition. The isRecord() method verifies that the query or query condition returns a Service Manager record.

Syntax

```
SCFile.isRecord();
```

Arguments

There are no arguments for this method.

Return values

A Boolean value: 1 (true) or 0 (false).
The method returns a Boolean value of 1 if the File object contains a Service Manager record or returns 0 if the method does not find a Service Manager record.
This method *does not* use the Service Manager-defined global return codes.

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Displays the incident record object and the results of the isRecord() method

This example requires the following sample data:

- A valid incident record (for example, "IM1005")
- An invalid incident record (for example "incident")

```
var incidentQuery;
```

```
function findTypes( query )
{
 print( "Searching for incident records starting with " + query +
"..." );
 var incidentFile = new File( "probsummary", "number#\""+ query +
"\"" );
 if ( incidentFile != null )
 {
  print( "Successfully allocated a File object");
  if ( incidentFile.isRecord() )
  {
   print( "Success. Found incident records starting with " + query +
". The first record is:\n" + incidentFile );
   return incidentFile
  }
  else
  {
   print( "Could not find incident records starting with " + query +
"." );
   return null
  }
 }
 else
 {
  print( "Could not allocate a File object." );
  return null
 }
}

incidentQuery = "IM1005";
findTypes( incidentQuery );

incidentQuery = "incident";
findTypes( incidentQuery );
```

JavaScript method: SCFile.join()

This method returns the elements of either a Service Manager Datum array object or a JavaScript array. This method does not use the Service Manager global return code values.

Syntax

```
SCFile.join();
```

Arguments

There are no arguments for this method.

Return values

A string.

The method returns the elements of an array.

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Displays the new incident record as a text string
- Displays the contents of action field using the join() method

This example requires the following sample data:

- A valid value for the number field (for example, "IM1010")
- An incident record with an action field value (for example, "IM1010")

```
var incidentID;

function findIncident( id )
{
 print( "Searching for Incident record: " + id + "..." );
 var incidentFile = new SCFile( "probsummary" );
 var rc = incidentFile.doSelect( "number=\"" + id + "\"" )
 if ( rc == RC_SUCCESS )
 {
  print( "Success. found Incident record:\n" + incidentFile.getText
() );
  print( "Displaying the contents of the action field as an object:
" + incidentFile.action );
  print( "Displaying the contents of the action field with join()
method: " + incidentFile.action.join() );
  return incidentFile
 }
 else
 {
  print( "Could not find Incident record. " + RCtoString( rc ) );
  return null
 }
}

incidentID = "IM1010";
findIncident( incidentID );
```

JavaScript method: SCFile.JSDate()

This method is deprecated as of version 6.1 with the introduction of the SCFile object. This method converted Service Manager date/time data into JavaScript date/time objects. The SCFile object now automatically converts Service Manager date/time data into JavaScript date/time objects. Alternatively, you can now use the XMLDate object and its methods to convert between date/time formats.

Syntax

```
SCFile.JSDate( dataType );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
dataType	String	This argument specifies the data type of a field in an SCFile object.

Return values

None

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Displays the incident record and the value of the open.time field
- Creates an SCDatum object containing the number of milliseconds you want to convert
- Converts the open.time field into an SCDatum object (by default the open.time field data is in Service Manager's date/time data type)
- Converts the SCDatum object into a JavaScript date/time

This example requires the following sample data:

- A valid incident record with a value in the open.time field (for example, "IM1010")

```
var incidentID;

function findIncident( id )
{
 print( "Searching for incident: " + id + "..." );
 var incidentFile = new SCFile( "probsummary" );
 var f = incidentFile.doSelect( "number=\""+ id + "\"" );
 if ( f == RC_SUCCESS )
 {
  print( "Success. found " + id + " in incident record:\n" +
incidentFile.getText() );
  print( "The value of the open.time field is: " +
incidentFile.open_time );
  print( "Converting open.time to an SCDatum object..." );
  var dat = new SCDatum( incidentFile.open_time );
  print( "The new SCDatum object is: " + dat );
  print( "Converting the SCDatum object to a JSDate..." );
  var j = dat.JSDate();
  print( "The converted SCDatum object is: " + j );
  return incidentFile
 }
 else
 {
  print( "Could not find incident record. " + RCtoString( f ) );
  return null
 }
```

```
}

incidentID = "IM1010";
findIncident( incidentID );
```

JavaScript method: SCFile.length()

This method returns the number of array elements in an array stored in a Service Manager Datum object. You can use this method to return the length of Service Manager array fields, which are always stored as Datum objects. To query a particular array field in an SCFile object, use the following format: `SCFile_object.array_field_name.length()`.

Note: Replace any periods in the name of *array_field_name* with underscores. For example, you must convert the Service Manager array field *update.action* to *update_action* when querying it from JavaScript.

Syntax

```
SCFile.array_field_name.length();
```

Arguments

There are no arguments for this method.

Return values

A number and RC_SUCCESS or one of the other global return code values.
The method returns the number of array elements in an array stored in a Service Manager Datum object and a global return code value of RC_SUCCESS or returns one of the error global return code values if the method cannot return the number of array elements.

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Attempts to get the size of the returned SCFile object (the result is always zero)
- Displays the contents of the action field (the action field contains Service Manager array data)
- Displays the size of the action field using the Service Manager length() method
- Converts the SCFile object into a JavaScript array using the Service Manager toArray() method
- Displays the size of the new JavaScript array using the core JavaScript length property

```
var incidentQuery;

function findIncidents( query )
{
 print( "Searching for incident records starting with " + query +
"..." );
 var incidentFile = new SCFile( "probsummary" );
 var findIncident = incidentFile.doSelect( "number#\""+ query + "\""
```

```
);
 if ( findIncident == RC_SUCCESS )
 {
  print( "Success. Found incident records starting with " + query +
". The first record is:\n" + incidentFile );
  var objectSize = incidentFile.getSize();
  print( "Running the getSize() method on the returned SCFile object
produces the following result: " + objectSize );
  var actionText = incidentFile.action;
  print( "The contents of the action field is:\n" + actionText );
  var actionSize = incidentFile.action.getSize();
  print( "The size of the action field is: " + actionSize );
  print( "Converting the SCFile object to a JavaScript array..." );
  var incidentArray = incidentFile.toArray();
  print( "The new JavaScript array is:\n" + incidentArray );
  var arraySize = incidentArray.length;
  print( "The size of the converted array is: " + arraySize );
       return incidentFile;
 }
 else
 {
  print( "Could not find incident records starting with " + query +
". " + RCtoString( findIncident ) );
  return null
 }
}

incidentQuery = "IM1005";
findIncidents( incidentQuery );
```

JavaScript method: SCFile.pop()

This method removes the last element from a JavaScript array and returns the element removed as a string. This method does not use the Service Manager global return code values.

Syntax

```
SCFile.pop();
```

Arguments

There are no arguments for this method.

Return values

A string.

The method returns the element removed from the array.

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Displays the new incident record as a text string

- Converts the action field into a JavaScript array
- Adds an item to the end of the array and displays the total number of elements in the array
- Removes an item from the end of the array and displays the array item removed

This example requires the following sample data:

- A valid value for the number field (for example, "IM1010")
- An incident record with an action field value (for example, "IM1010")

```
var incidentID;

function findIncident( id )
{
 print( "Searching for Incident record: " + id + "..." );
 var incidentFile = new SCFile( "probsummary" );
 var rc = incidentFile.doSelect( "number=\"" + id + "\"" )
 if ( rc == RC_SUCCESS )
 {
  print( "Success. found Incident record:\n" + incidentFile.getText
() );
  print( "Converting the action field to a JavaScript array..." );
  var a = incidentFile.action.toArray();
  print( "The action field array contains the following: " + a );
  print( "Pushing new array entry..." );
  var addOne = a.push( "First array item" );
  print( "The number of items in the array are: " + addOne );
  print( "The action field array contains the following: " + a );
  print( "Popping new array entry..." );
  var removeOne = a.pop();
  print( "The array item removed was: " + removeOne );
  print( "The action field array contains the following: " + a );
  return incidentFile;
 }
 else
 {
  print( "Could not find Incident record. " + RCtoString( rc ) );
  return null
 }
}

incidentID = "IM1010";
findIncident( incidentID );
```

JavaScript method: SCFile.push()

This method adds an entry to the end of a JavaScript array and returns the total number of entries in the array. If you apply this method to a Service Manager array object, then it adds the item but does

not return the number of entries in the array. This method does not use the Service Manager global return code values.

Syntax

```
SCFile.push( arrayItem );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
arrayItem	String	This argument specifies the array item you want to add to a JavaScript array.

Return values

A number.

The method returns the number of items in the array.

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Displays the new incident record as a text string
- Converts the contents of the action field into a JavaScript array
- Adds an item to the end of the JavaScript array and prints the total number of items in the array

This example requires the following sample data:

- A valid value for the number field (for example, "IM1010")
- An incident record with an action field value (for example, "IM1010")

```
var incidentID;

function findIncident( id )
{
 print( "Searching for Incident record: " + id + "..." );
 var incidentFile = new SCFile( "probsummary" );
 var rc = incidentFile.doSelect( "number=\"" + id + "\"" )
 if ( rc == RC_SUCCESS )
 {
  print( "Success. found Incident record:\n" + incidentFile.getText
() );
  print( "Converting the action field to a JavaScript array..." );
  var a = incidentFile.action.toArray();
  print( "The action field array contains the following: " + a );
  print( "Pushing new array entry..." );
  var addOne = a.push( "First array item" );
  print( "The number of items in the array are: " + addOne );
  print( "The action field array contains the following: " + a );
  print( "Pushing new array entry..." );
  var addTwo = a.push( "Second array item" );
```

```
    print( "The number of items in the array are: " + addTwo );
    print( "The action field array contains the following: " + a );
    return incidentFile;
   }
   else
   {
    print( "Could not find Incident record. " + RCtoString( rc ) );
    return null
   }
  }

  incidentID = "IM1010";
  findIncident( incidentID );
```

JavaScript method: SCFile.setFields()

This method causes any subsequent doSelect() calls to fetch only the specified fields. If setFields has not been called on a table, all fields are selected and a “SELECT *” is performed.

This method is supported for read-only objects. You can make an object read-only specifying the SCFILE_READONLY argument in the SCFile constructor.

Calling setFields with no parameters will reset any previously specified fields. Any subsequent calls to doSelect() will fetch all fields (a SELECT * will be performed).

Note: Do not reference fields in the object that have not been specified in the setFields method. If you reference field in the object that has not been specified in the setFields method, the field appears as “null” whether the database has data in that column or not.

Syntax

SCFile.setFields(*field-array*);

SCFile.setFields(*field-string*);

SCFile.setFields( );

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
field-array	Array	A javascript array of field name strings; for example, setFields(new Array(“contact.name”, “user.id”)).
field-string	String	A string of space-separated field names; for example, setFields (“contact.name user.id”)

Return values

The method returns RC_SUCCESS if successful and RC_ERROR on failure.

The following conditions cause a setFields failure:

- Calling setFields on an uninitialized SCFile or a non read-only SCFile
- Calling setFields with an invalid argument type (not a string or an array of strings)
- Calling setFields with one (or more) invalid fields. (The method performs a basic validation on the field names to ensure they exist in the DBDICT.)

When the method fails, it has no effect, and the last successful call to setFields stays in effect. It issues a warning in the sm.log file.

Limitations

- The main unique key fields of the dbdict are always selected from the table (and for join tables, from each underlying table) even if not explicitly specified in setFields.
- Since only basic field validation is performed in the setFields method, there are certain situations when the call can be successful (returns RC_SUCCESS) but a “SELECT *” is still performed. This would be the case if:
  - All specified fields exist in the DBDICT but at least one field is an alias field, a structure name or an unmapped field (which are unsupported).
  - LDAP is the primary data source for the table. (LDAP mapped fields are supported as long as LDAP is not the primary data source.)

Example usage

This example does the following:

- Takes the prefix of a contact name and returns a list of contacts that start with the specified prefix.
- On failure, returns the message "Could not find contact."

This example requires the following sample data:

- a contacts table with some records

```
function findPrefix( prefix )
{
   var contactList = new SCFile( "contacts", SCFILE_READONLY );
//added new parameter SCFILE_READONLY
   var fields = new Array("contact.name", "user.id"); // build array
of fields to be returned from DB
   var rc = contactList.setFields( fields ); // call new setFields
method to identify fields to query from DB

   if ( rc != RC_SUCCESS )
   {
      print( "setFields() failed, will revert to SELECT * (see log
for more info)" );
      return null;
   }

   var findContact = contactList.doSelect( "contact.name # \"" +
```

```
prefix + "\"" );
   if ( findContact == RC_SUCCESS )
   {
      return contactList;
   }
   else
   {
      print( "Could not find contact. " + RCtoString( findContact )
);
      return null;
   }
}
```

JavaScript method: SCFile.setOrderBy()

This method causes any subsequent doSelect() call on a SCFile object to fetch only the specified fields by the specified sort sequences of the SCFileObject. If setOrderBy() has not been called, a subsequent doSelect() will use the default sort sequence.

Syntax

```
SCFile.setOrderBy(sortFields, sortSeqs);
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
sortFields	Array	A javascript array of field name string, for example, newArray ("name","company");
sortSeqs	Array	A javascript array of sort sequenence specification. It is either a SCFILE_ASC, or SCFILE_DSC. for example : new Array(SCFILE_ASC, SCFILE_DSC); SCFILE_ASC is a constant for specifying sort by ascending order, and SCFILE_DSC for specifying sort by descending order.

Required properties

This method requires two parameters of the data type of array. The two arrays must be of the same length. Calling setOrderBy() with parameters of uneven length of arrays or non-array data results in failure return status.

Return values

Returns RC_SUCCESS if successful and RC_ERROR on failure.
The following conditions cause a setOrderBy() failure:

- Calling setOrderBy() on an uninitialized SCFile
- Calling setOrderBy() with an invalid argument type (not an array) for either argument.
- Calling setOrderBy() with wrong number of arguments.
- Calling setOrderBy() with two arrays of different length.

- Calling setOrderBy() with one (or more) invalid fields. (The method performs a basic validation on the field names to ensure they exist in the DBDICT.

A warning is written to SM.log file.

If the sort sequence specifications in the sortSeq array is of non-recognizable constant, SCFILE_ASC is assumed.

Limitations:

• Fields used in setOrderBy() should not be within an array in the dbdict.

• Fields from a join file should not be within a structure in the dbdict.

Example

Note: If you are running setOrderBy() on a single array, these samples show how to define a single sort sequence for descending sort order.

1. `var sortOrder = [SCFILE_DSC];`
2. `var sortOrder = new Array(1);`
   `sortOrder[0] = SCFILE_DSC;`

This example does the following:

- Takes two fields, name and company, from the operator table, and sorts the company field by descending order, then the name field by ascending order.
- On failure returns a RC_ERROR status code.

This example requires the following sample data:

- Records in the operator table that have various company and name fields.

```
function runOrderByTest( table, fields, sortOrder)
{
  var timeBefore = new Date();
  var fCount = 0;
  print( "doSelect select column OrderBy test : Start time is: " +
timeBefore );
  var fRec = new SCFile( table);
  var setOrderByRC = fRec.setOrderBy( fields, sortOrder);

  if ( setOrderByRC != RC_SUCCESS )
  {
    print( "ERROR: setOrderBy failed, RC=" + RCtoString(
setOrderByRC ) );
    return RC_ERROR;
  }
  var selectRC = fRec.doSelect( "true" );
  if ( selectRC != RC_SUCCESS )
  {
    print( "ERROR: doSelect OrderBy failed, RC=" + RCtoString(
selectRC ) );
    return RC_ERROR;
  }
```

```
  do
  {
    fCount++;
  }
  while (fRec.getNext() == RC_SUCCESS);

  var timeAfter = new Date();
  print( "doSelect select column Orderby test: table processed = " +
table );
  print( "doSelect select column Orderby test: fields processed = "
+ fields );
  print( "doSelect select column Orderby test: sort order = " +
sortOrder);
  print( "doSelect select column Orderby test: count = " + fCount );
  print( "doSelect select column Orderby test: End time is " +
timeAfter );
  print( "doSelect select column Orderby test: Elapsed time is " +
(timeAfter - timeBefore) + " milliseconds" );

  return RC_SUCCESS;
}

var table = "operator";
var fields = new Array("company", "name");
var sortOrder = new Array(SCFILE_DSC, SCFILE_ASC);
var testRC = runOrderByTest( table, fields, sortOrder);
```

JavaScript method: SCFile.setType()

This method is deprecated as of ServiceCenter version 6.1 with the introduction of the SCFile object. This method set the data type of Service Manager datum objects. The SCFile object now automatically converts Service Manager data types into corresponding JavaScript data types. You can also use the XMLDate object and its methods to convert between date/time formats.

Syntax

```
SCFile.setType( dataType );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
dataType	Number	This argument specifies the Service Manager data type for the object. See "Data types" on page 20.

Return values

None

Example

This example does the following:

- Creates an SCDatum object containing the number of milliseconds you want to convert
- Sets the data type of the SCDatum object to a Service Manager date/time type
- Converts the number of milliseconds into a JavaScript date/time

This example requires the following sample data:

- A number of milliseconds (for example, "1234567890")

```
var millisecs;

function findDateTime( num )
{
 var t = new SCDatum( num );
 t.setType( 3 );
 print( num + " milliseconds is equal to the date: " + t.JSDate() );
}

millisecs = 1234567890;
findDateTime( millisecs );
```

JavaScript method: scfile_setrecord

Sets the SCFile field values from a given XML string.

Syntax

```
SCFile.setRecord();
```

Argument

setRecord("xmlstring");

Example

```
function updateContactPhoneFromXML( name, xmlstr )
{
    print("Searching for contact: "+ name + "...");
    var contactRecord = new SCFile("contacts");
    var rc = contactRecord.doSelect( "contact.name=\""+ name + "\""
);
    if ( rc == RC_SUCCESS)
    {
        print("found contact " + name + ", phone number:\n" +
contactRecord.contact_phone.toString() );
        contactRecord.setRecord(xmlstr);
        contactRecord.doUpdate();
        print("updated phone number in contact record:\n" +
contactRecord.contact_phone.toString());
    }
    else
    {
        print("Could not find contact. " + RCtoString( rc ) );
```

```
    }
}

var contactName = "ADMINISTRATOR, SYSTEM";
var xmlString = "<model name=\"contacts\"
query=\"contact.name="ADMINISTRATOR, SYSTEM"\"><keys><contact.name
sctype=\"string\">ADMINISTRATOR,
SYSTEM</contact.name></keys><instance recordid=\"ADMINISTRATOR,
SYSTEM\" uniquequery=\"contact.name="ADMINISTRATOR,
SYSTEM"\"><contact.name mandatory=\"true\"
type=\"string\">ADMINISTRATOR, SYSTEM</contact.name><contact.phone
type=\"string\">(335) 123-4567</contact.phone></instance></model>";

updateContactPhoneFromXML(contactName, xmlString);
```

Notes

- The input XML string should follow the Service Manager File schema.
- You cannot use setRecord() to set values for files that contain binary data. For example,you cannot use setRecord() for displaycache, code, SYSATTACHMENTS files.
- You cannot use setRecord() to set dbdict records.

JavaScript method: SCFile.setValue()

This method updates an HP Service Manager record with the new field value defined in the *newValue* argument. This method does not use the Service Manager global return codes.

Syntax

```
SCFile.setValue( newValue );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
newValue	String	This argument specifies the new value you want the Service Manager object to have. You must use the slash character to escape out quotation marks and any special characters restricted from JavaScript.

Return values

None

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Displays the current value of the action field
- Sets the value of the action field to any value you define in the input variable

This example requires the following sample data:

- A valid incident record with a value in the action field (for example, "IM1010")
- A valid action field value update (for example, set the action field to "New description")

```
var incidentID;
var newValue;

function findIncident( id, value )
{
 print( "Searching for Incident record: " + id + "..." );
 var incidentFile = new SCFile( "probsummary" );
 var rc = incidentFile.doSelect( "number=\"" + id + "\"" )
 if ( rc == RC_SUCCESS )
 {
  print( "Success. found Incident record " + id );
  print( "Displaying the contents of the action field as an object:
" + incidentFile.action );
  print( "Setting the value of the action field..." );
  incidentFile.action.setValue( value );
  print( "Displaying the contents of the action field as an object:
" + incidentFile.action );
  print( "Displaying the Incident record: " + incidentFile );
  return incidentFile
 }
 else
 {
  print( "Could not find Incident record. " + RCtoString( rc ) );
  return null
 }
}

incidentID = "IM1010";
newValue = "New description";
findIncident( incidentID, newValue );
```

JavaScript method: SCFile.shift()

This method removes the first element from a JavaScript array and returns the element removed as a string. This method does not use the Service Manager global return code values.

Syntax

```
SCFile.shift();
```

Arguments

There are no arguments for this method.

Return values

A string.

The method returns the element removed from the array.

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Displays the new incident record as a text string
- Converts the action field into a JavaScript array
- Adds an item to the end of the array and displays the total number of elements in the array
- Removes an item from the beginning of the array and displays the array item removed

This example requires the following sample data:

- A valid value for the number field (for example, "IM1010")
- An incident record with an action field value (for example, "IM1010")

```
var incidentID;

function findIncident( id )
{
 print( "Searching for Incident record: " + id + "..." );
 var incidentFile = new SCFile( "probsummary" );
 var rc = incidentFile.doSelect( "number=\"" + id + "\"" )
 if ( rc == RC_SUCCESS )
 {
  print( "Success. found Incident record:\n" + incidentFile.getText
() );
  print( "Converting the action field to a JavaScript array..." );
  var a = incidentFile.action.toArray();
  print( "The action field array contains the following: " + a );
  print( "Pushing new array entry..." );
  var addOne = a.push( "First array item" );
  print( "The number of items in the array are: " + addOne );
  print( "The action field array contains the following: " + a );
  print( "Shifting array one entry..." );
  var removeOne = a.shift();
  print( "The array item removed was: " + removeOne );
  print( "The action field array contains the following: " + a );
  return incidentFile;
 }
 else
 {
  print( "Could not find Incident record. " + RCtoString( rc ) );
  return null
 }
}

incidentID = "IM1010";
findIncident( incidentID );
```

JavaScript method: SCFile.toArray()

This method converts an HP Service Manager Datum object into a JavaScript array.

Syntax

```
SCFile.toArray();
```

Arguments

There are no arguments for this method.

Return values

A JavaScript array and RC_SUCCESS or one of the other global return code values.

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Displays the incident record as a text string
- Converts the contents of the action field into a JavaScript array

This example requires the following sample data:

- A valid incident record number (for example, "IM1010")
- An incident record with an action field value (for example, "IM1010")

```
var incidentID;

function findIncident( id )
{
 print( "Searching for Incident record: " + id + "..." );
 var incidentFile = new SCFile( "probsummary" );
 var rc = incidentFile.doSelect( "number=\"" + id + "\"" )
 if ( rc == RC_SUCCESS )
 {
  print( "Success. found Incident record:\n" + incidentFile.getText
() );
  print( "Converting the action field to a JavaScript array..." );
  var a = incidentFile.action.toArray();
  print( "The action field contains the following " + a );
  return incidentFile
 }
 else
 {
  print( "Could not find Incident record. " + RCtoString( rc ) );
  return null
 }
}

incidentID = "IM1010";
findIncident( incidentID );
```

JavaScript method: SCFile.unshift()

This method adds an entry to the beginning of a JavaScript array and returns the total number of entries in the array. If you apply this method to a Service Manager array object, then it adds the item but does not return the number of entries in the array. This method does not use the Service Manager global return code values.

Syntax

```
SCFile.unshift( arrayItem );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
arrayItem	String	This argument specifies the array item you want to add to a JavaScript array.

Return values

A number.

The method returns the number of items in the array.

Example

This example does the following:

- Searches the probsummary table for any incident record you define in the search variable
- Displays the new incident record as a text string
- Converts the contents of the action field into a JavaScript array
- Adds an item to the beginning of the JavaScript array and prints the total number of items in the array

This example requires the following sample data:

- A valid value for the number field (for example, "IM1010")
- An incident record with an action field value (for example, "IM1010")

```
var incidentID;

function findIncident( id )
{
 print( "Searching for Incident record: " + id + "..." );
 var incidentFile = new SCFile( "probsummary" );
 var rc = incidentFile.doSelect( "number=\"" + id + "\"" )
 if ( rc == RC_SUCCESS )
 {
  print( "Success. found Incident record:\n" + incidentFile.getText
() );
  print( "Converting the action field to a JavaScript array..." );
  var a = incidentFile.action.toArray();
  print( "The action field array contains the following: " + a );
  print( "Unshifting array one entry..." );
```

```
    var unshifting = a.unshift( "First array item" );
    print( "The number of items in the array are: " + unshifting );
    print( "The action field array contains the following: " + a );
    return incidentFile;
   }
   else
   {
    print( "Could not find Incident record. " + RCtoString( rc ) );
    return null
   }
  }

  incidentID = "IM1010";
  findIncident( incidentID );
```

JavaScript method: SCFile.updateAttachment(attachObj)

This method updates a specific attachment associated with the current file record. The attachment object must contain a valid attachment ID in the "href" property of the attachment object, as well as valid values for the other required properties (name, type, and value). Before calling updateAttachment, you must establish a current record using one of the positioning methods, such as doSelect(), getNext(), etc.

Syntax

```
SCFile.updateAttachment( attachObj );
```

Arguments

updateAttachment() takes one argument, which is an Attachment object.

Return values

If successful, updateAttachment() returns a string of the form cid:xxxxxx containing the new attachment ID for the attachment. Otherwise, it returns NULL.

Example

This example does the following:

- Selects one record from the probsummary table
- Gets a specific attachment associated with the current record
- Updates the attachment

This example requires the following sample data:

- "IM10001" record in "probsummary" table
- An attachment associated with "IM10001" record

```
function createTestAttachment()
```

```
{
  var attachmentObj = new Attachment();

  attachmentObj.type  = "text/plain";
  attachmentObj.name  = "MyAttachment.txt";
  attachmentObj.value = "My text attachment";

  var f = new SCFile( 'probsummary' );

  var rc = f.doSelect( 'number = "IM10001"' );

  var attachmentID = f.insertAttachment( attachmentObj );

  return attachmentID;
}

var attachmentID = createTestAttachment();

var f = new SCFile( 'probsummary' );
var rc = f.doSelect( 'number = "IM10001"' );
var attachmentObj = f.getAttachment( attachmentID );

// Now update the attachment object

var newName = "NewName.text";
var newValue = "Updated attachment text value";

attachmentObj.name = newName;
attachmentObj.value = newValue;

// Do the update. This should result in a new attachment ID

var newID =  f.updateAttachment( attachmentObj );
```

JavaScript object: SCRecordList

The following JavaScript object is unique to HP Service Manager.

SCRecordList is an array of Datums that qualify with the query.

Constructor

```
new SCRecordList( <table name>, <QueryCond object>);
```

Arguments

The following arguments are valid for this object:

Argument	Data type	Description
table name	String	The name of the Service Manager table; for example,

		probsummary.
QueryCond	object	Service Manager JavaScript object QueryCond which defines the SC query to be executed.

Properties

None

Methods

The following methods are valid for this object:

Defined method	Description
getCount()	This method returns the total number of records in the SCRecordList.
getPosition()	This method returns the last position in the record list that has been accessed. .

Example

This example does the following:

- lists the location's position in a record list
- counts the records in the list

This example requires the following sample data:

- valid location(s)
- valid record list with at least one record

```
var x = new SCRecordList( 'location', new QueryCond('location.name',
LIKE, 'a'));

print("Total record in the list: " + x.getCount());

for (i in x ) {
  print(x[i].location_name + " is the " + x.getPosition() + " record
in the record list." );
}
```

JavaScript method: SCRecordList.getCount()

This method counts the total number of records in the SCRecordList.

Syntax

```
SCRecordList.getCount();
```

Arguments

There are no arguments for this method.

Return values

This method returns the total number of records in a record list.

Example

This example does the following:

- counts the records in the list

This example requires the following sample data:

- valid location(s)
- valid record list with at least one record

```
var x = new SCRecordList( 'location', new QueryCond('location.name',
LIKE, 'a'));

print("Total record in the list: " + x.getCount());
```

JavaScript method: SCRecordList.getPosition()

This method returns the position of the last accessed record in the record list. If there are no records are in the SCRecordList, it returns -1. If there is at least one record and it has not been accessed, the position is set to the first element, which is position 0.

Syntax

```
SCRecordList.getPosition();
```

Arguments

There are no arguments for this method.

Return values

Last position of Datum accessed in record list.
Returns -1 if there are no records in the list.
Returns 0 if list is not yet or the first Datum is the last Datum accessed and the list contains at least one record.

Example

This example does the following:

- lists the location's position in a record list
- counts the records in the list

This example requires the following sample data:

- valid location(s)
- valid record list with at least one record

```
var x = new SCRecordList( 'location', new QueryCond('location.name',
LIKE, 'a'));

print("Total record in the list: " + x.getCount());

for (i in x ) {
  print(x[i].location_name + " is the " + x.getPosition() + " record
in the record list." );
}
```

JavaScript object: XML

The following JavaScript object is unique to HP Service Manager.

The XML() constructor creates an empty XML object where you can store and manipulate XML documents.

This object's methods *do not* use the Service Manager-defined global return codes.

Constructor

```
new XML();
```

Arguments

None

Properties

None

Methods

The following methods are valid for this object:

Defined method	Description
setContent	Creates an XML document from a string or file.
getParentNode	Returns an XML object representing the parent node of the current node.
getFirstChildElement	Returns an XML object representing the first child node of the current node.
getNextSiblingElement	Returns an XML object representing the next node at the same level in the Document Object Model (DOM) tree as the current node.
getFirstAttribute	Returns an XML object representing the first attribute of the current node.
getNextAttribute	Returns an XML object representing the next attribute of the current node.
getNodeName	Returns a string representing the name of the current element or attribute.
getName	Returns a string representing the name of the current element or attribute.
getQualifiedName	Returns a string representing the name of the current element or attribute including any namespace value.
getPrefix	Returns a string representing the namespace value of the current element or attribute.
getNodeValue	Returns a string representing the value of the current element.
getValue	Returns a string representing the value of the current element.

setNodeValue	Adds or updates the value of the current element.
setValue	Adds or updates the value of the current element.
getNodeType	Evaluates an XML object and returns an integer representing the XML DOM type of the current element or attribute.
isDocumentElement	Returns true if the current element is the DOM document element.
getDocumentElement	Creates an XML object containing the document element and all child elements from an XML object or XML string.
createNode	Creates an XML node from an XML object.
appendNode	Inserts an XML node into an XML object.
importNode	Copies an XML node from one XML document to another.
addAttribute	Inserts an XML attribute and attribute value into the current element.
getAttributeNode	Returns an XML object containing an attribute node.
getAttributeValue	Returns the value of the target attribute.
setAttributeValue	Adds or updates an attribute value.
addElement	Inserts an XML element as a child of the current element.
setText	Adds or updates the value of the current element.
getText	Returns a string representing the value of the current element.
toXMLString	Converts an XML object into a valid XML string.

Example

This example does the following:

- Creates an XML object
- Sets the content of the XML object to a local XML file

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")

```
/* Instantiate an empty XML object */
var xmlObj = new XML();

/* Instantiate a variable containing the path to an XML file */
var xmlFile = "C:\\myxmldoc.xml";

/* Use setContent to make the XML object read and parse the XML file
*  If the setContent method fails, print an error message */
```

```
if ( ! xmlObj.setContent( xmlFile, true ) )
{
  print( "setContent failed. Unable to parse xml: " + xmlFile );
}
```

JavaScript method: XML.addAttribute()

This method inserts an XML attribute and attribute value into the current element. It requires you use the other XML get methods to navigate through an XML document.

Syntax

```
XML.addAttribute( AttributeName, AttributeValue );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
AttributeName	String	This argument specifies the text string you want the script to use as the XML attribute name. The string argument must contain characters valid for an XML element (for example, the string cannot include the characters < or >).
AttributeValue	String	This argument specifies the text string you want the script to use as the XML attribute value. This argument must contain characters valid for an XML element (for example, the string cannot include the characters < or >).

Return values

An XML object or null.

If successful, the method returns the updated XML element. Otherwise, it returns null.

Example

This example adds a new attribute and attribute value to any element you select from an XML document.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test02.xml" containing "<document><parent1>value1</parent1><parent2></parent2></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test02.xml", true );
var XMLSource;
var TargetNode;
var searchResults;
```

```
var printResults;
var AttributeName;
var AttributeValue;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}

/* Create a function to print the target element and search results
*/
function createNewAttribute( searchResult, targetNode, attrName,
attrValue )
{
 print( "The target element we are looking for is: " + targetNode );
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
```

```
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  print( "Adding new attribute " + attrName + " to element " +
elementName );
  searchResult.addAttribute( attrName, attrValue );
  print( "The new object is:\n" + searchResult );
  return searchResult
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}

/* Set variable values for XMLFile and TargetNode. Run functions.
*/
print( "Processing XML from XMLFile... \n" );
TargetNode = "parent1"
XMLSource = XMLFile
searchResults = findTargetElement( XMLSource, TargetNode );
AttributeName = "id";
AttributeValue = "test";
createNewAttribute( searchResults, TargetNode, AttributeName,
AttributeValue );
```

JavaScript method: XML.addElement()

This method inserts an XML element as a child of the current element. You must use the other XML get methods to navigate through an XML document.

Syntax

```
XML.addElement( String );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
String	String	This argument specifies the text string you want the script to use as the XML element name. The string argument must contain characters valid for an XML element (for example, the string cannot include the characters < or >).

Return values

An XML object or null.

The method returns an XML object containing the new element or returns null if the method cannot add the element.

Example

This example adds a child element to any element you select in an XML document.
This example requires the following sample data:

- A local XML file (for example, a file called "C:\test02.xml" containing "<document><parent1>value1</parent1><parent2></parent2></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test02.xml", true );
var XMLSource;
var TargetNode;
var searchResults;
var printResults;
var NewNodeName;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
```

```
    }
   }
   return node;
  }

  /* Create a function to print the target element and search results
  */
  function createNewNode( searchResult, targetNode, nodeName )
  {
   print( "The target element we are looking for is: " + targetNode );
   if ( searchResult != null )
   {
    var elementName = searchResult.getNodeName();
    print( "Found element " + elementName + " in: \n" + searchResult +
  "\n" );
    print( "Adding new node " + nodeName + " to element " +
  elementName );
    var newNode = searchResult.addElement( nodeName );
    print( "The new object is:\n" + searchResult );
    return searchResult
   }
   else
   {
    print( "Cannot find " + target );
    return null
   }
  }

  /* Set variable values for XMLFile and TargetNode. Run functions.
  */
  print( "Processing XML from XMLFile... \n" );
  TargetNode = "parent1"
  XMLSource = XMLFile
  searchResults = findTargetElement( XMLSource, TargetNode );
  NewNodeName = "child8";
  createNewNode( searchResults, TargetNode, NewNodeName );
```

JavaScript method: XML.appendNode()

This method inserts an XML node into an XML object. There must be an existing XML object with a document element. This method does not update the existing XML object.

Syntax

```
XML.appendNode( name );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
name	String	This argument specifies the XML node name you want to append to the target XML object. The string argument must contain characters valid for XML (for example, the string cannot include the characters < or > except as XML entities such as < and >).

Return values

An XML object or null.

The method returns an XML object containing the new node or returns null if the method cannot append the node to the current target node.

Example

This example appends an XML node to an XML source object.

This example requires the following sample data:

- An XML object (for example, an object containing "<document> </document>")

```
var XMLObject = new XML;
var NewNodeName;
var NewNodeValue;
var NewNodeType;
var TargetNode;
XMLObject = XMLObject.setContent( "<document> </document>" );

function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
```

```
}

function appendNewNode( type, name, value, xmlSource, target )
{
 print( "The XML source object is: " + xmlSource);
 print( "Creating node of type: " + type + " named: " + name + "
with value: " + value );
 var NewNode = xmlSource.createNode( type, name, value );
 print( "Appending " + NewNode.toXMLString() + " to: " + target );
 var FoundNode = findTargetElement( xmlSource, target );
 var appendedNode = FoundNode.appendNode( NewNode );
 print( "The new XML object is: " +  xmlSource );
}

NewNodeType = 1;
NewNodeName = "node";
NewNodeValue = "test";
TargetNode = "document";
appendNewNode( NewNodeType, NewNodeName, NewNodeValue, XMLObject,
TargetNode );
```

JavaScript method: XML.createNode()

This method creates an XML node from an XML object. It requires an existing XML object with a document element, but it does not update the existing XML object. The new method requires a rootname.

Syntax

```
XML.createNode( type, name, value );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
type	Integer	This argument specifies the XML DOM node type for the node you want the script to create. This method only accepts XML DOM types 1, 2, and 3. See the W3C Web site for a complete listing of XML DOM node-types by integer.
name	String	This argument specifies the text string you want the script to add as the XML node name. The string argument must contain characters valid for XML (for example, the string cannot include the characters < or > except as XML entities such as < and >).
value	String	This argument specifies the text string you want the script to add as the XML node value. The string argument must contain characters valid for XML (for example, the string cannot include the characters < or > except as XML entities such as < and >).

Return values

An XML object or null.

The method returns an XML object containing the new node or returns null if the method cannot create the node.

Example

This example creates an XML node from an XML source object.

This example requires the following sample data:

- An XML object (for example, an object containing "<document> </document>")

```
var XMLObject = new XML("document");
var NewNodeName;
var NewNodeValue;
var NewNodeType;
XMLObject = XMLObject.setContent( "<document> </document>" );

function createNewNode( type, name, value, xmlSource )
{
 print( "The XML source object is: " + xmlSource);
 print( "Creating node of type: " + type + " named: " + name + "
with value: " + value );
 var NewNode = xmlSource.createNode( type, name, value );
 print( "The XML source object remains unchanged: " + XMLObject );
 print( "The new node converted to an XML string is: " +
NewNode.toXMLString() );
 print( "Applying the getName method to the new node produces: " +
NewNode.getName() );
 print( "Applying the getValue method on the new node produces: " +
NewNode.getValue() );
}

NewNodeType = 1;
NewNodeName = "node";
NewNodeValue = "test";
createNewNode( NewNodeType, NewNodeName, NewNodeValue, XMLObject );

NewNodeType = 2;
NewNodeName = "id";
NewNodeValue = "test";
createNewNode( NewNodeType, NewNodeName, NewNodeValue, XMLObject );

NewNodeType = 3;
NewNodeName = null;
NewNodeValue = "Text node";
createNewNode( NewNodeType, NewNodeName, NewNodeValue, XMLObject );
```

JavaScript method: XML.getAttributeNode()

This method returns an XML object containing the target attribute, or returns null if the method cannot find the target attribute in the current element. You must use the other XML get methods to navigate through an XML document.

Syntax

```
XML.getAttributeNode( Attribute );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
Attribute	String	This argument specifies the XML attribute you want the script to return.

Return values

An XML object or null.

The method returns an XML object containing the target attribute, or returns null if the method cannot find the target attribute in the current element.

Example

This example searches for a particular attribute of any given element in an XML document. This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLFile to store a valid XML file.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test.xml", true );
var XMLSource;
var TargetNode;
var TargetAttribute;
var searchResults;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
```

```
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}

/* Create a function to print the target element and search results
*/
function printFindResults( target, searchResult, attribute )
{
 print( "The target element we are looking for is: " + target );
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  var attributeFound = findAttributeNode( searchResult, attribute );
  return attributeFound
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}

/* Create a function to find the attribute node of the target
element */
function findAttributeNode( sourceObject, attribute )
{
```

```
 print( "Looking for attribute node " + attribute );
 var attributeNode = sourceObject.getAttributeNode( attribute );
 if ( attributeNode != null )
 {
  print( "Success. Found attribute " + attributeNode );
  return attributeNode
 }
 else
 {
  print( "Fail. Did not find attribute " + attribute );
  return null
 }
}

/* Set variable values for XMLFile and TargetNode. Run functions  */
print( "Processing XML from XMLFile... \n" );
TargetNode = "child1";
XMLSource = XMLFile;
TargetAttribute = "attribute1";
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults, TargetAttribute );
```

JavaScript method: XML.getAttributeValue()

This method returns the value of the target attribute. You must use the other XML get methods to navigate through an XML document.

Syntax

```
XML.getAttributeValue( Attribute );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
Attribute	String	This argument specifies the XML attribute whose value you want the script to return.

Return values

A string or null.

The method returns the string value of the target attribute or returns null if the method cannot find an attribute value.

Example

This example searches for the attribute value of any given element in an XML document.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLFile to store a valid XML file.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test.xml", true );
var XMLSource;
var TargetNode;
var TargetAttribute;
var searchResults;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}

/* Create a function to print the target element and search results
*/
function printFindResults( target, searchResult, attribute )
```

```
{
 print( "The target element we are looking for is: " + target );
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  var attributeFound = findAttributeValue( searchResult, attribute
);
  return attributeFound
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}

/* Create a function to find the attribute node of the target
element */
function findAttributeValue( sourceObject, attribute )
{
 print( "Looking for attribute value of " + attribute );
 var attributeValue = sourceObject.getAttributeValue( attribute );
 if ( attributeValue != null )
 {
  print( "Success. The value of " + attribute + " is " +
attributeValue );
  return attributeValue
 }
 else
 {
  print( "Fail. Did not find attribute " + attribute );
  return null
 }
}

/* Set variable values for XMLFile and TargetNode. Run functions  */
print( "Processing XML from XMLFile... \n" );
TargetNode = "child1";
XMLSource = XMLFile;
TargetAttribute = "attribute1";
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults, TargetAttribute );
```

JavaScript method: XML.getDocumentElement()

This method searches an XML object for the DOM document element (or "root" node). It creates an XML object containing the document element and all child elements from an XML object or XML string.

Note: You cannot use this method on objects that you have converted using the toXMLString method.

Syntaz

```
XML.getDocumentElement();
```

Arguments

There are no arguments for this method.

Return values

An XML object or null.
If successful, this method returns an XML object containing the document element and all of its child elements. This method returns null if applied to an empty or invalid XML object.

Example

This example displays the XML document element from four different input sources.
This example requires the following sample data:

- A valid XML object (for example, the list of currently logged on users stored in system.users)
- A valid XML string (for example, "<document><parent1><child1 attribute1=\"1\" /></parent1><parent2 /></document>")
- A noncompliant XML string (for example, "<document1><parent1 /></document1><document2><parent2 /></document2>")
- An empty XML object

```
/* Create variables to store XML objects and strings.
*  Use goodXML to store a valid XML string.
*  Use badXML to store any erroneous XML string, in this case a
second document element.
*  Use xmlObject to store a valid XML object such as the
system.users list
*  The script sets the content of xmlString to goodXML.
*  The script sets the content of xmlBad to badXML.
*  The script sets the content of xmlEmpty to an XML object. */
var goodXML = "<document><parent1><child1 attribute1=\"1\"
/></parent1><parent2 /></document>"
var badXML = "<document1><parent1 /></document1><document2><parent2
/></document2>"
var xmlObject = system.users
var xmlString = new XML();
xmlString = xmlString.setContent( goodXML );
var xmlBad = new XML();
xmlBad = xmlBad.setContent( badXML );
print( "The value of xmlBad is:\n" + xmlBad );
```

```
var xmlEmpty = new XML();

/* Create a function to execute the getDocumentElement method on the
document argument. */
function getDocElem( document )
{
 var DocElem = document.getDocumentElement();
 if ( DocElem != null )
 {
  var DocElemName = DocElem.getNodeName();
  print( "Success. Found the document element " + DocElemName + " in
the following object: \n" + document );
  return DocElem
 }
 else
 {
  print( "Error. Did not find document element in the following
object: \n" + document );
  return null
 }
}

/* Run the getDocElem and getDocElemName functions on four input
sources.
*  Print the objects returned by each function. */
print( "Now searching for a document element in a valid XML
object...\n" );
getDocElem( xmlObject );
print( "Now searching for a document element in a valid XML
string...\n" );
getDocElem( xmlString );
print( "Now searching for a document element in a non-compliant XML
string...\n" );
getDocElem( xmlBad );
print( "Now searching for a document element in an empty XML
object...\n" );
getDocElem( xmlEmpty );
```

JavaScript method: XML.getFirstAttribute()

This method returns an XML object representing the first attribute of the current node.

Syntax

```
XML.getFirstAttribute();
getFirstAttribute(Element)
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
Element	String	This argument contains the name of the element you want the method to use as the starting position when searching for the first attribute.

Return values

An XML object or null.

The method returns an object representing the first attribute of the current node or returns null if the XML object has no attributes.

Example

This example displays the first attribute of any element you select. The example allows you to select an element from two different input sources.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")
- A valid XML string (for example, "<document><parent1><child1 /><child2 attribute2=\"b\" /></parent1><parent2 /></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test.xml", true );
var sourceXMLString = "<document><parent1><child1 /><child2
attribute2=\"b\" /></parent1><parent2 /></document>";
var XMLString = new XML();
XMLString = XMLString.setContent( sourceXMLString );
var XMLSource;
var TargetNode;
var searchResults;
var attributeResults;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
```

```
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}

/* Create a function to print the target element and search results
*/
function printFindResults( target, searchResult )
{
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  return searchResult
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}

/* Create a function to find the first attribute of the target
element */
function findFirstAttribute( sourceObject, targetNode )
{
 var startingNode = findTargetElement( sourceObject, targetNode )
 var attribute = startingNode.getFirstAttribute();
 if ( attribute != null )
 {
```

```
  return attribute
 }
 else
 {
  return null
 }
}

/* Create a function to print the first attribute of the target
element */
function printAttributeResults( target, searchResult )
{
 if ( searchResult != null )
 {
  var attributeName = searchResult.getNodeName();
  print( "The first attribute of " + target + " is: " +
attributeName );
 }
 else
 {
  print( "There are no attributes of " + target );
 }
}

/* Set variable values for XMLFile and TargetNode. Run functions  */
print( "Processing XML from XMLFile... \n" );
TargetNode = "child1"
XMLSource = XMLFile
print( "The target element we are looking for is: " + TargetNode );
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults );
attributeResults = findFirstAttribute( XMLSource, TargetNode );
printAttributeResults( TargetNode, attributeResults );

/* Set variable values for XMLFile and TargetNode. Run functions  */
print( "Processing XML from XMLString... \n" );
TargetNode = "child2"
XMLSource = XMLString
print( "The target element we are looking for is: " + TargetNode );
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults );
attributeResults = findFirstAttribute( XMLSource, TargetNode );
printAttributeResults( TargetNode, attributeResults );
```

JavaScript method: XML.getFirstChildElement()

This method returns an XML object representing the first child node of the current node.

Syntax

```
XML.getFirstChildElement();
getFirstChildElement(Element)
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
Element	String	This argument contains the name of the element you want the method to use as the starting position when searching for the first child element.

Return values

An XML object or null.

The method returns an object representing the first child node of the current node or returns null if the XML object has no child node.

Example

This example displays the first child element of any element you select. The example allows you to select an element from two different input sources.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")
- A valid XML string (for example, "<document><parent1><child1 /><child2 attribute2=\"b\" /></parent1><parent2 /></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test.xml", true );
var sourceXMLString = "<document><parent1><child1 /><child2
attribute2=\"b\" /></parent1><parent2 /></document>";
var XMLString = new XML();
XMLString = XMLString.setContent( sourceXMLString );
var XMLSource;
var TargetNode;
var searchResults;
var childResults;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
```

```
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}

/* Create a function to print the target element and search results
*/
function printFindResults( target, searchResult )
{
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  return searchResult
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}

/* Create a function to find the first child element of the target
element */
```

```
function findFirstChild( sourceObject, targetNode )
{
 var startingNode = findTargetElement( sourceObject, targetNode )
 var firstChild = startingNode.getFirstChildElement();
 if ( firstChild != null )
 {
  return firstChild
 }
 else
 {
  return null
 }
}

/* Create a function to print the first child element of the target
element */
function printChildResults( target, searchResult )
{
 if ( searchResult != null )
 {
  var childName = searchResult.getNodeName();
  print( "The first child element of " + target + " is: " +
childName );
 }
 else
 {
  print( "There are no child elements of " + target );
 }
}

/* Set variable values for XMLFile and TargetNode. Run functions  */
print( "Processing XML from XMLFile... \n" );
TargetNode = "parent1"
XMLSource = XMLFile
print( "The target element we are looking for is: " + TargetNode );
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults );
childResults = findFirstChild( XMLSource, TargetNode );
printChildResults( TargetNode, childResults );

/* Set variable values for XMLFile and TargetNode. Run functions  */
print( "Processing XML from XMLString... \n" );
TargetNode = "parent2"
XMLSource = XMLString
print( "The target element we are looking for is: " + TargetNode );
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults );
childResults = findFirstChild( XMLSource, TargetNode );
printChildResults( TargetNode, childResults );
```

JavaScript method: XML.getName()

This method returns a string representing the name of the current element or attribute. You must use the other XML get methods to navigate through an XML document. This method is a synonym for getNodeName();.

Syntax

```
XML.getName();
```

Arguments

There are no arguments for this method.

Return values

A string or null.

The method returns a string representing the name of the current element or attribute. It returns null if the current node has no name.

Example

This example displays the names of the elements and attributes in an XML document.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")
- A valid XML string (for example, "<document><parent1><child1>0123</child1></parent1></document>"")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test.xml", true );
var sourceXMLString =
"<document><parent1><child1>0123</child1></parent1></document>";
var XMLString = new XML();
XMLString = XMLString.setContent( sourceXMLString );
var XMLSource;

function walkXML( elem )
{
 print( "Element " + elem.getName() );
 var node = elem.getFirstAttribute();
 while( node != null )
 {
   var attrName  = node.getName();
   var attrValue = node.getNodeValue();
   print( "Attribute " + attrName + " = " + attrValue );
   node = elem.getNextAttribute();
 }
```

```
 var nodeValue = elem.getNodeValue();
 if ( typeof nodeValue == "string" && nodeValue.length > 0 )
 {
  print( "Element value of " + elem.getName() + " is: " + nodeValue
);
 }

 var child = elem.getFirstChildElement();
 while( child != null )
 {
   var nodeName  = child.getName();
   var nodeValue = child.getNodeValue();
   walkXML( child );
   child = child.getNextSiblingElement();
 }
}

print( "Now printing the structure of an XML file...\n" );
XMLSource = XMLFile
walkXML( XMLSource );

print( "Now printing the structure of an XML string...\n" );
XMLSource = XMLString
walkXML( XMLSource );
```

JavaScript method: XML.getNextAttribute()

This method returns an XML object representing the next attribute of the current node.

Syntax

```
XML.getNextAttribute();
getNextAttribute(Element)
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
Element	String	This argument contains the name of the element you want the method to use as the starting position when searching for the first attribute.

Return values

An XML object or null.

The method returns an object representing the first attribute of the current node or returns null if the XML object has no attributes.

Example

This example displays the first attribute of any element you select. The example allows you to select an element from two different input sources.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")
- A valid XML string (for example, "<document><parent1><child1 /><child2 attribute2=\"b\" attribute3=\"c\" /></parent1><parent2 /></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test.xml", true );
var sourceXMLString = "<document><parent1><child1 /><child2
attribute2=\"b\" attribute3=\"c\" /></parent1><parent2
/></document>";
var XMLString = new XML();
XMLString = XMLString.setContent( sourceXMLString );
var XMLSource;
var TargetNode;
var searchResults;
var attributeResults;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
```

```
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}

/* Create a function to print the target element and search results
*/
function printFindResults( target, searchResult )
{
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  return searchResult
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}

/* Create a function to find the first attribute of the target
element */
function findFirstAttribute( sourceObject, targetNode )
{
 var startingNode = findTargetElement( sourceObject, targetNode )
 var firstAttribute = startingNode.getFirstAttribute();
 if ( firstAttribute != null )
 {
  return firstAttribute
 }
 else
 {
  return null
 }
}

/* Create a function to print the first attribute of the target
element */
function printAttributeResults( target, searchResult )
{
```

```
 if ( searchResult != null )
 {
  var attributeName = searchResult.getNodeName();
  print( "The first attribute of " + target + " is: " +
attributeName );
  return searchResult
 }
 else
 {
  print( "There are no attributes of " + target );
  return null
 }
}

/* Create a function to find the next attribute of the target
element */
function findNextAttribute( sourceObject, targetNode )
{
 var startingNode = findTargetElement( sourceObject, targetNode )
 var firstAttribute = startingNode.getFirstAttribute();
 if ( firstAttribute != null )
 {
  var nextAttribute = startingNode.getNextAttribute();
  while ( nextAttribute != null )
  {
   var attributeName = nextAttribute.getNodeName();
   print( "The next attribute of " + targetNode + " is: " +
attributeName );
   nextAttribute = nextAttribute.getNextAttribute();
  }
  print( "There are no more attributes of " + targetNode );
  return targetNode
 }
 else
 {
  return null
 }
}

/* Set variable values for XMLFile and TargetNode. Run functions  */
print( "Processing XML from XMLFile... \n" );
TargetNode = "child1"
XMLSource = XMLFile
print( "The target element we are looking for is: " + TargetNode );
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults );
attributeResults = findFirstAttribute( XMLSource, TargetNode );
printAttributeResults( TargetNode, attributeResults );
findNextAttribute( XMLSource, TargetNode );
```

```
/* Set variable values for XMLFile and TargetNode. Run functions  */
print( "Processing XML from XMLString... \n" );
TargetNode = "child2"
XMLSource = XMLString
print( "The target element we are looking for is: " + TargetNode );
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults );
attributeResults = findFirstAttribute( XMLSource, TargetNode );
printAttributeResults( TargetNode, attributeResults );
findNextAttribute( XMLSource, TargetNode );
```

JavaScript method: XML.getNextSiblingElement()

This method returns an XML object representing the next node at the same level in the Document Object Model (DOM) tree as the current node.

Syntax

```
XML.getNextSiblingElement();
getNextSiblingElement(Element)
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
Element	String	This argument contains the name of the element you want the method to use as the starting position when searching for the next sibling element.

Return values

An XML object or null.

The method returns an object representing the next node at the same level in the DOM tree as the current node or returns null if the XML object has no sibling node.

Example

This example displays the next sibling element of any element you select. The example allows you to select an element from two different input sources.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")
- A valid XML string (for example, "<document><parent1><child1 /><child2 attribute2=\"b\" /></parent1><parent2 /></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
```

```
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test.xml", true );
var sourceXMLString = "<document><parent1><child1 /><child2
attribute2=\"b\" /></parent1><parent2 /></document>";
var XMLString = new XML();
XMLString = XMLString.setContent( sourceXMLString );
var XMLSource;
var TargetNode;
var searchResults;
var siblingResults;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}

/* Create a function to print the target element and search results
```

```
*/
function printFindResults( target, searchResult )
{
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  return searchResult
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}

/* Create a function to find the next sibling element of the target
element */
function findNextSibling( sourceObject, targetNode )
{
 var startingNode = findTargetElement( sourceObject, targetNode )
 var nextSibling = startingNode.getNextSiblingElement();
 if ( nextSibling != null )
 {
  return nextSibling
 }
 else
 {
  return null
 }
}

/* Create a function to print the first attribute of the target
element */
function printSiblingResults( target, searchResult )
{
 if ( searchResult != null )
 {
  var childName = searchResult.getNodeName();
  print( "The next sibling element of " + target + " is: " +
childName );
 }
 else
 {
  print( "There are no sibling elements of " + target );
 }
}

/* Set variables to search the XMLFile for the TargetNode.
```

```
*  Run functions findTargetElement, printFindResults, findAttribute,
and
*  printAttributeResults on the XMLFile and TargetNode */
print( "Processing XML from XMLFile... \n" );
TargetNode = "child3"
XMLSource = XMLFile
print( "The target element we are looking for is: " + TargetNode );
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults );
siblingResults = findNextSibling( XMLSource, TargetNode );
printSiblingResults( TargetNode, siblingResults );

/* Set variables to search the XMLString for the TargetNode.
*  Run functions findTargetElement, printFindResults, findAttribute,
and
*  printAttributeResults on the XMLString and TargetNode */
print( "Processing XML from XMLString... \n" );
TargetNode = "child1"
XMLSource = XMLString
print( "The target element we are looking for is: " + TargetNode );
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults );
siblingResults = findNextSibling( XMLSource, TargetNode );
printSiblingResults( TargetNode, siblingResults );
```

JavaScript method: XML.getNodeName()

This method returns a string representing the name of the current element or attribute. You must use the other XML get methods to navigate through an XML document.

Syntax

```
XML.getNodeName();
```

Arguments

There are no arguments for this method.

Return values

A string or null.

The method returns a string representing the name of the current element or attribute or returns null if the current node has no name.

Example

This example displays the names of the elements and attributes in an XML document.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")

- A valid XML string (for example, "<document><parent1><child1>0123</child1></parent1></document>"")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test.xml", true );
var sourceXMLString =
"<document><parent1><child1>0123</child1></parent1></document>";
var XMLString = new XML();
XMLString = XMLString.setContent( sourceXMLString );
var XMLSource;

function walkXML( elem )
{
 print( "Element " + elem.getNodeName() );
 var node = elem.getFirstAttribute();
 while( node != null )
 {
   var attrName  = node.getNodeName();
   var attrValue = node.getNodeValue();
   print( "Attribute " + attrName + " = " + attrValue );
   node = elem.getNextAttribute();
 }
 var nodeValue = elem.getNodeValue();
 if ( typeof nodeValue == "string" && nodeValue.length > 0 )
 {
  print( "Element value of " + elem.getNodeName() + " is: " +
nodeValue );
 }

 var child = elem.getFirstChildElement();
 while( child != null )
 {
   var nodeName  = child.getNodeName();
   var nodeValue = child.getNodeValue();
   walkXML( child );
   child = child.getNextSiblingElement();
 }
}

print( "Now printing the structure of an XML file...\n" );
XMLSource = XMLFile
walkXML( XMLSource );

print( "Now printing the structure of an XML string...\n" );
XMLSource = XMLString
walkXML( XMLSource );
```

JavaScript method: XML.getNodeType()

This method evaluates an XML object and returns an integer representing the XML DOM type of the current element or attribute. You must use the other XML get methods to navigate through an XML document.

Syntax

```
XML.getNodeType();
```

Arguments

There are no arguments for this method.

Return values

A string or null.

The method returns an integer representing the XML DOM type of the current element or attribute or returns null if the method cannot determine the XML DOM type. See the W3C Web site for a complete listing of XML DOM node types by integer (www.w3.org/). It returns a TypeError if applied to anything other than an XML object. This limits this method to elements and attributes since the Service Manager XML object cannot represent element values as an XML object.

Example

This example displays the XML DOM types of the elements and attributes in an XML document. This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1>value1<child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")

```
function visitElement( elem )
{
 print( "Element " + elem.getQualifiedName() + " is of type " +
elem.getNodeType() );
 /* Print attributes for this element */
 var node = elem.getFirstAttribute();
 while( node != null )
 {
   var attrName  = node.getQualifiedName();
   var attrValue = node.getNodeValue();
   print( "Attribute " + attrName + "=" + attrValue + " and is of
type " + node.getNodeType() );
   node = elem.getNextAttribute();
 }
 /* Print possible element value */
 var nodeValue = elem.getNodeValue();
 if ( typeof nodeValue == "string" && nodeValue.length > 0 )
 {
    print( "Element value is " + nodeValue + " and is of type " +
```

```
elem.getNodeType() );
 }
 /* Now print child elements */
 var child = elem.getFirstChildElement();
 while( child != null )
 {
   var nodeName  = child.getNodeName();
   var nodeValue = child.getNodeValue();
   visitElement( child );
   child = child.getNextSiblingElement();
 }
}

var xmlObj =  new XML();
xmlObj.setContent( "c:\\test.xml", true );
visitElement( xmlObj.getDocumentElement() );
```

JavaScript method: XML.getNodeValue()

This method returns a string representing the value of the current element. It requires you use the other XML get methods to navigate through an XML document.

Syntax

```
XML.getNodeValue();
```

Arguments

There are no arguments for this method.

Return values

A string or null.

The method returns a string representing the value of the current XML element or returns null if the current node has no value.

Example

This example displays the value of the elements in an XML document.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test02.xml" containing "<document><parent1>value1</parent1><parent2></parent2></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test02.xml", true );
var XMLSource;
var TargetNode;
```

```
/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}

/* Create a function to print the target element and search results
*/
function printFindResults( searchResult, targetNode )
{
 print( "The target element we are looking for is: " + targetNode );
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  findValue( searchResult, targetNode );
  return searchResult
 }
```

```
 else
 {
  print( "Cannot find " + target );
  return null
 }
}

/* Create a function to find the parent node element of the target
element */
function findValue( sourceObject, targetNode )
{
 var valueFound = sourceObject.getNodeValue();
 if ( valueFound.length >= 1 )
 {
  print( "The value of " + targetNode + " is: " + valueFound );
  return valueFound
 }
 else
 {
  print( "There is no value for " + targetNode );
  valueFound = null
  return valueFound
 }
}

/* Set variable values for XMLFile and TargetNode. Run functions.
*/
print( "Processing XML from XMLFile... \n" );
TargetNode = "parent1"
XMLSource = XMLFile
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( searchResults, TargetNode );

/* Set variable values for XMLFile and TargetNode. Run functions.
*/
print( "Processing XML from XMLFile... \n" );
TargetNode = "parent2"
XMLSource = XMLFile
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( searchResults, TargetNode );
```

JavaScript method: XML.getParentNode()

This method returns an XML object representing the parent node of the current node.

Syntax

```
XML.getParentNode();
```

```
getParentNode(Element)
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
Node	String	This argument contains the name of the element you want the method to use as the starting position when searching for the parent node element.

Return values

An XML object or null.

The method returns an object representing the parent node of the current node or returns null if the XML object has no parent node.

Example

This example displays the parent node element of any element you select. The example allows you to select an element from two different input sources.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")
- A valid XML string (for example, "<document><parent1><child1 /><child2 attribute2=\"b\" /></parent1><parent2 /></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test.xml", true );
var sourceXMLString = "<document><parent1><child1 /><child2
attribute2=\"b\" /></parent1><parent2 /></document>";
var XMLString = new XML();
XMLString = XMLString.setContent( sourceXMLString );
var XMLSource;
var TargetNode;
var searchResults;
var parentResults;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
```

```
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}

/* Create a function to print the target element and search results
*/
function printFindResults( target, searchResult )
{
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  return searchResult
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}

/* Create a function to find the parent node of the target element
*/
function findParent( sourceObject, targetNode )
{
 var startingNode = findTargetElement( sourceObject, targetNode )
```

```
 var parentNode = startingNode.getParentNode();
 if ( parentNode != null )
 {
  return parentNode
 }
 else
 {
  return null
 }
}

/* Create a function to print the first attribute of the target
element */
function printParentResults( target, searchResult )
{
 if ( searchResult != null )
 {
  var parentName = searchResult.getNodeName();
  print( "The parent node element of " + target + " is: " +
parentName );
 }
 else
 {
  print( "There is no parent node elements of " + target );
 }
}

/* Set variables to search the XMLFile for the TargetNode.
*  Run functions findTargetElement, printFindResults, findAttribute,
and
*  printAttributeResults on the XMLFile and TargetNode */
print( "Processing XML from XMLFile... \n" );
TargetNode = "child3"
XMLSource = XMLFile
print( "The target element we are looking for is: " + TargetNode );
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults );
parentResults = findParent( XMLSource, TargetNode );
printParentResults( TargetNode, parentResults );

/* Set variables to search the XMLString for the TargetNode.
*  Run functions findTargetElement, printFindResults, findAttribute,
and
*  printAttributeResults on the XMLString and TargetNode */
print( "Processing XML from XMLString... \n" );
TargetNode = "child1"
XMLSource = XMLString
print( "The target element we are looking for is: " + TargetNode );
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults );
```

```
parentResults = findParent( XMLSource, TargetNode );
printParentResults( TargetNode, parentResults );
```

JavaScript method: XML.getPrefix()

This method returns a string representing the namespace value of the current element or attribute. It requires you use the other XML get methods to navigate through an XML document.

Syntax

```
XML.getPrefix();
```

Arguments

There are no arguments for this method.

Return values

A string or null.

The method returns a string representing the namespace value for the current node (element or attribute) or returns null if the current node has no namespace value.

Example

This example displays the namespace value of the elements and attributes in an XML document.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\namespace.xml" containing <?xml version="1.0" encoding="UTF-8" standalone="yes" ?><document xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:ns="http://servicemanager.hp.com/PWS"><http:element1 /><ns:element2 /></document>

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\namespace.xml", true );
var XMLSource;
var TargetNode;
var searchResults;
var prefixResults;
var firstAttribute;
var attributeResults;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
```

```
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}

/* Create a function to print the target element and search results
*/
function printFindResults( searchResult, targetNode )
{
 print( "The target element we are looking for is: " + targetNode );
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  findPrefix( searchResult, targetNode );
  return searchResult
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}
```

```
/* Create a function to find the parent node element of the target
element */
function findPrefix( sourceObject, targetNode )
{
 var prefixFound = sourceObject.getPrefix();
 if ( prefixFound.length >= 1 )
 {
  print( "The namespace prefix of " + targetNode + " is: " +
prefixFound );
  return prefixFound
 }
 else
 {
  print( "There is no namespace prefix of " + targetNode );
  prefixFound = null
  return prefixFound
 }
}

/* Create a function to find the first attribute of the target
element */
function findFirstAttribute( sourceObject, targetNode )
{
 var startingNode = findTargetElement( sourceObject, targetNode )
 var firstAttribute = startingNode.getFirstAttribute();
 if ( firstAttribute != null )
 {
  var attributeName = firstAttribute.getNodeName();
  print( "The first attribute of " + targetNode + " is: " +
attributeName );
  findPrefix( firstAttribute, targetNode );
  return firstAttribute
 }
 else
 {
  print( "There are no attributes of " + targetNode );
  return null
 }
}

/* Create a function to find the next attribute of the target
element */
function findNextAttribute( sourceObject, targetNode )
{
 var startingNode = findTargetElement( sourceObject, targetNode )
 var firstAttribute = startingNode.getFirstAttribute();
 if ( firstAttribute != null )
 {
  var nextAttribute = startingNode.getNextAttribute();
```

```
  while ( nextAttribute != null )
  {
   var attributeName = nextAttribute.getNodeName();
   print( "The next attribute of " + targetNode + " is: " +
attributeName );
   findPrefix( nextAttribute, targetNode );
   nextAttribute = nextAttribute.getNextAttribute();
  }
  print( "There are no more attributes of " + targetNode );
  return targetNode
 }
 else
 {
  return null
 }
}

/* Set variable values for XMLFile and TargetNode. Run functions.
*/
print( "Processing XML from XMLFile... \n" );
TargetNode = "element1"
XMLSource = XMLFile
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( searchResults, TargetNode );

/* Set variable values for XMLFile and TargetNode. Run functions.
*/
print( "Processing XML from XMLFile... \n" );
TargetNode = "document"
XMLSource = XMLFile
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( searchResults, TargetNode );
findFirstAttribute( XMLSource, TargetNode );
findNextAttribute( XMLSource, TargetNode );
```

JavaScript method: XML.getQualifiedName()

This method returns a string representing the name of the current element or attribute, including any namespace value. You must use the other XML get methods to navigate through an XML document.

Syntax

```
XML.getQualifiedName();
```

Arguments

There are no arguments for this method.

Return values

A string or null.

The method returns a string representing the name of the current element or attribute. It returns null if the current node has no name.

Example

This example displays the qualified names of the elements and attributes in an XML document. This example requires the following sample data:

- A local XML file (for example, a file called "C:\namespace.xml" containing "<?xml version="1.0" encoding="UTF-8" standalone="yes" ?><document xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:ns="http://servicemanager.hp.com/PWS"><http:element1 /><ns:element2 /></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\namespace.xml", true );
var XMLSource;

function nsXMLWalk( elem )
{
 print( "Element " + elem.getQualifiedName() );
 var node = elem.getFirstAttribute();
 while( node != null )
 {
   var attrName  = node.getQualifiedName();
   var attrValue = node.getNodeValue();
   print( "Attribute " + attrName + " = " + attrValue );
   node = elem.getNextAttribute();
 }
 var nodeValue = elem.getNodeValue();
 if ( typeof nodeValue == "string" && nodeValue.length > 0 )
 {
  print( "Element value of " + elem.getQualifiedName() + " is: " +
nodeValue );
 }

 var child = elem.getFirstChildElement();
 while( child != null )
 {
   var nodeName  = child.getQualifiedName();
   var nodeValue = child.getNodeValue();
   nsXMLWalk( child );
   child = child.getNextSiblingElement();
 }
}

function walkXML( elem )
{
 print( "Element " + elem.getName() );
 var node = elem.getFirstAttribute();
```

```
 while( node != null )
 {
   var attrName  = node.getName();
   var attrValue = node.getNodeValue();
   print( "Attribute " + attrName + " = " + attrValue );
   node = elem.getNextAttribute();
 }
 var nodeValue = elem.getNodeValue();
 if ( typeof nodeValue == "string" && nodeValue.length > 0 )
 {
  print( "Element value of " + elem.getName() + " is: " + nodeValue
);
 }

 var child = elem.getFirstChildElement();
 while( child != null )
 {
   var nodeName  = child.getName();
   var nodeValue = child.getNodeValue();
   walkXML( child );
   child = child.getNextSiblingElement();
 }
}

print( "Now printing the structure of an XML file with qualified
names...\n" );
XMLSource = XMLFile
nsXMLWalk( XMLSource );

print( "Now printing the structure of an XML file without qualified
names...\n" );
XMLSource = XMLFile
walkXML( XMLSource );
```

JavaScript method: XML.getText()

This method returns a string representing the value of the current element. You must use the other XML get methods to navigate through an XML document. This method is a synonym for getNodeValue();.

Syntax

```
XML.getText();
```

Arguments

There are no arguments for this method.

Return values

A string or null.

The method returns a string representing the value of the current XML element or returns null if the current node has no value.

Example

This example displays the value of the elements in an XML document.
This example requires the following sample data:

- A local XML file (for example, a file called "C:\test02.xml" containing "<document><parent1>value1</parent1><parent2></parent2></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test02.xml", true );
var XMLSource;
var TargetNode;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
```

```
  }
 }
 return node;
}

/* Create a function to print the target element and search results
*/
function printFindResults( searchResult, targetNode )
{
 print( "The target element we are looking for is: " + targetNode );
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  findValue( searchResult, targetNode );
  return searchResult
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}

/* Create a function to find the parent node element of the target
element */
function findValue( sourceObject, targetNode )
{
 var valueFound = sourceObject.getText();
 if ( valueFound.length >= 1 )
 {
  print( "The value of " + targetNode + " is: " + valueFound );
  return valueFound
 }
 else
 {
  print( "There is no value for " + targetNode );
  valueFound = null
  return valueFound
 }
}

/* Set variable values for XMLFile and TargetNode. Run functions.
*/
print( "Processing XML from XMLFile... \n" );
TargetNode = "parent1"
XMLSource = XMLFile
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( searchResults, TargetNode );
```

```
/* Set variable values for XMLFile and TargetNode. Run functions.
*/
print( "Processing XML from XMLFile... \n" );
TargetNode = "parent2"
XMLSource = XMLFile
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( searchResults, TargetNode );
```

JavaScript method: XML.getValue()

This method returns a string representing the value of the current element. You must use the other XML get methods to navigate through an XML document. This method is a synonym for getNodeValue();.

Syntax

```
XML.getValue();
```

Arguments

There are no arguments for this method.

Return values

A string or null.

The method returns a string representing the value of the current XML element or returns null if the current node has no value.

Example

This example displays the value of the elements in an XML document.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test02.xml" containing "<document><parent1>value1</parent1><parent2></parent2></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test02.xml", true );
var XMLSource;
var TargetNode;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
```

```
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}

/* Create a function to print the target element and search results
*/
function printFindResults( searchResult, targetNode )
{
 print( "The target element we are looking for is: " + targetNode );
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  findValue( searchResult, targetNode );
  return searchResult
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}
```

```
/* Create a function to find the parent node element of the target
element */
function findValue( sourceObject, targetNode )
{
 var valueFound = sourceObject.getValue();
 if ( valueFound.length >= 1 )
 {
  print( "The value of " + targetNode + " is: " + valueFound );
  return valueFound
 }
 else
 {
  print( "There is no value for " + targetNode );
  valueFound = null
  return valueFound
 }
}

/* Set variable values for XMLFile and TargetNode. Run functions.
*/
print( "Processing XML from XMLFile... \n" );
TargetNode = "parent1"
XMLSource = XMLFile
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( searchResults, TargetNode );

/* Set variable values for XMLFile and TargetNode. Run functions.
*/
print( "Processing XML from XMLFile... \n" );
TargetNode = "parent2"
XMLSource = XMLFile
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( searchResults, TargetNode );
```

JavaScript method: XML.importNode()

This method copies an XML node from one XML document to another. It requires two existing XML objects with document elements: one is the source XML object and the other is the target XML document. This method does not update either the source or the target XML object. Instead, you must use the appendNode method to add the imported node into the target XML document.

Syntax

```
XML.importNode( name );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description

name	String	This argument specifies the XML node name you want to copy from the source XML object. The string argument must contain characters valid for XML (for example, the string cannot include the characters < or > except as XML entities such as < and >).

Return values

An XML object or null.

The method returns an XML object containing the copied node or returns null if the method cannot import a copy of the node.

Example

This example imports an XML node from one XML object to another.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")
- An XML object (for example, an object containing "<document> </document>")

```
var XMLFile = new XML();
var XMLObject = new XML();
var TargetNode;

function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}
```

```
/* Create a function to print the target element and search results
*/
function printFindResults( target, searchResult )
{
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  return searchResult
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}

function importNewNode( xmlSource, xmlTarget, node )
{
 print( "The XML source object is:\n" + xmlSource );
 print( "The target element we are looking for is: " + node );
 var FoundNode = findTargetElement( xmlSource, node );
 printFindResults( node, FoundNode );
 print( "The XML target object is:\n" + xmlTarget );
 print( "Importing " + FoundNode + " into XML target object..." );
 var NewNode = xmlTarget.importNode( FoundNode );
 print( "The XML target object remains unchanged: " +  xmlTarget );
 print( "Appending " + FoundNode + " into XML target object...");
 var appendedNode = xmlTarget.appendNode( NewNode );
 print( "The new XML target object is: " +  xmlTarget );
}

XMLFile = XMLFile.setContent( "C:\\test.xml", true );
XMLObject = XMLObject.setContent( "<document> </document>" );
TargetNode = "parent1";
importNewNode( XMLFile, XMLObject, TargetNode );
```

JavaScript method: XML.isDocumentElement()

This method returns true if the current element is the Document Object Model (DOM) document element (the "root" node of the XML document). You must use the other XML get methods to navigate through an XML document.

Syntax

```
XML.isDocumentElement();
```

Arguments

There are no arguments for this method.

Return values

A Boolean value: true or false.
The method returns true if the current XML element is the DOM document element or false if the current XML element is not the DOM document .

Example

This example checks to see if the selected element is the DOM document element.
This example requires the following sample data:

- A local XML file (for example, a file called "C:\test02.xml" containing "<document><parent1>value1</parent1><parent2></parent2></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test02.xml", true );
var XMLSource;
var TargetNode;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
```

```
    }
    else
    {
     node = childNode;
    }
   }
   return node;
  }

  /* Create a function to print the target element and search results
  */
  function printFindResults( searchResult, targetNode )
  {
   print( "The target element we are looking for is: " + targetNode );
   if ( searchResult != null )
   {
    var elementName = searchResult.getNodeName();
    print( "Found element " + elementName + " in: \n" + searchResult +
  "\n" );
    testDocElem( searchResult);
    return searchResult
   }
   else
   {
    print( "Cannot find " + target );
    return null
   }
  }

  /* Create a function to find the parent node element of the target
  element */
  function testDocElem( sourceObject )
  {
   var docElement = sourceObject.isDocumentElement();
   if ( docElement == true )
   {
    var nodeName = sourceObject.getName();
    print( "Success. The element " + nodeName + " is the DOM document
  element." );
    return valueFound
   }
   else
   {
    var nodeName = sourceObject.getName();
    print( "Fail. The element " + nodeName + " is not the DOM document
  element." );
    valueFound = null
    return valueFound
   }
  }
```

```
/* Set variable values for XMLFile and TargetNode. Run functions.
*/
print( "Processing XML from XMLFile... \n" );
TargetNode = "parent1"
XMLSource = XMLFile
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( searchResults, TargetNode );

/* Set variable values for XMLFile and TargetNode. Run functions.
*/
print( "Processing XML from XMLFile... \n" );
TargetNode = "document"
XMLSource = XMLFile
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( searchResults, TargetNode );
```

JavaScript method: XML.setAttributeValue()

This method adds or updates the attribute value of the target attribute. You must use the other XML get methods to navigate through an XML document.

Syntax

```
XML.setAttributeValue( AttributeName, AttributeValue );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
AttributeName	String	This argument specifies the XML attribute whose value you want the script to add or update.
AttributeValue	String	This argument specifies the text string you want the script to use as the XML attribute value. This argument must contain characters valid for an XML element (for example, the string cannot include the characters < or >).

Return values

This method does not return any values.

Example

This example adds or updates the attribute value of any given element in an XML document. This example requires the following sample data:

- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")

```
/* Create variables to store XML objects and strings.
```

```
*  Use XMLFile to store a valid XML file.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test.xml", true );
var XMLSource;
var TargetNode;
var TargetAttribute;
var TargetValue;
var searchResults;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */
function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}

/* Create a function to print the target element and search results
*/
function printFindResults( target, searchResult, attributeName,
```

```
attributeValue )
{
 print( "The target element we are looking for is: " + target );
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  var attributeSet = setAttributeValue( searchResult, attributeName,
attributeValue );
  print( "The XML object is " + searchResult );
  return attributeSet
 }
 else
 {
  print( "Cannot find " + target );
  return null
 }
}

/* Create a function to set the attribute value of the target
attribute */
function setAttributeValue( sourceObject, attributeName,
attributeValue )
{
 print( "Setting attribute value of " + attributeName );
 var setAttributeValue = sourceObject.setAttributeValue(
attributeName, attributeValue );
 var newAttributeValue = sourceObject.getAttributeValue(
attributeName );
 print( "The value of " + attributeName + " is now " +
newAttributeValue );
 return newAttributeValue
}

/* Set variable values for XMLFile and TargetNode. Run functions  */
print( "Processing XML from XMLFile... \n" );
TargetNode = "child1";
XMLSource = XMLFile;
TargetAttribute = "attribute1";
TargetValue = "z";
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults, TargetAttribute,
TargetValue );

print( "Processing XML from XMLFile... \n" );
TargetNode = "child2";
XMLSource = XMLFile;
TargetAttribute = "attribute1";
TargetValue = "z";
```

```
searchResults = findTargetElement( XMLSource, TargetNode );
printFindResults( TargetNode, searchResults, TargetAttribute,
TargetValue );
```

JavaScript method: XML.setContent()

This method creates an XML object from a string or the contents of an external file. It always creates a valid XML document with a document element. The method removes any values outside the document element.

Note: This method returns null if applied to an existing XML object because the header information ([C++ object XML] @nnnnnnnnnn -) is not valid.

Syntax

```
XML.setContent( String ); setContent( "FilePath", IsFile );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
String	String	This argument specifies the text string you want the script to parse as on XML object. The string argument must contain a valid XML document with a document element.
FilePath	String	This argument specifies the path to the XML document you want the script to parse an XML object. You must enclose the FilePath argument in quotation marks and use the true argument to indicate that the first argument is a file path and not a string value.
IsFile	Boolean	This argument specifies whether the first argument is a string or a file path. A true value indicates the first argument is a file path. Any other value, including omitting the second argument, indicates that the first argument is a string.

Return values

An XML object or null.

The method returns an XML object if it parses the XML input and returns null if it cannot parse the input.

Arguments

Use the toXMLString method to convert an XML object to an XML string.

Example

This example attempts to create an XML object from five different sources.

This example requires the following sample data:

- A valid XML string (for example, "<document><parent1><child1 attribute1=\"1\" /></parent1><parent2 /></document>")

- An XML string with more than one document element (for example, "<document1><parent1 /></document1><document2><parent2 /></document2>")
- A local XML file (for example, a file called "C:\test.xml" containing "<document><parent1><child1 attribute1="a" /><child2 /></parent1><parent2><child3 /><child4 /></parent2></document>")
- An existing XML object (for example, the list of currently logged on users stored in system.users)
- An XML object converted to a string (for example, the list of currently logged on users stored in system.users)

```
/* Create a function to execute the setContent method.
*  The function creates a new XML object xmlObj.
*  It sets the content of the XML object to the source argument.
*  The isFile argument determines if the source is an XML string or
file path.
*  If setContent returns true, it prints an XML string and returns
the XML object.
*  If setContent returns false, it prints the source and returns
null. */
function setXML( source, isFile )
{
 var xmlObj = new XML();

 if ( xmlObj.setContent( source, isFile ) )
 {
  var xmlDoc = xmlObj.toXMLString();
  print( "setContent succeeded! \n The XML document is: \n" + xmlDoc
);
  return xmlObj
 }
 else
 {
  print( "setContent failed. \n Could not parse the following input
as XML: \n" + source );
  return null
 }
}

/* Create variables to store your XML source objects.
*  Use xmlObject to test setting content to an XML object.
*  xmlObjectConverted converts xmlObject to an XML string.
*  Note: The toXMLstring method removes the object header
information.
*  Use xmlString to test setting content to an XML string.
*  Note: You must use the backslash character to escape out
quotation marks in your XML string.
*  Use xmlFile to test setting content to an XML file. */
var xmlString = "<document><parent1><child1 attribute1=\"1\"
/></parent1><parent2 /></document>";
```

```
var xmlBad = "<document1><parent1 /></document1><document2><parent2
/></document2>"
var xmlFile = "C:\\test.xml";
var xmlObject = system.users
var xmlObjectConverted = xmlObject.toXMLString();

/* Note: You can omit the isFile argument when the content source is
an XML string.
*  See the xmlString example below. */
print( "Testing setting content to a valid XML string...\n" );
setXML( xmlString );
print( "Testing setting content to an XML string with more than one
document element...\n" );
print( "The non-compliant XML string before conversion is: \n" +
xmlBad );
setXML( xmlBad );
print( "Testing setting content to an XML file...\n" );
setXML( xmlFile, true );
print( "Testing setting content to an existing XML object...\n" );
setXML( xmlObject, false );
print( "Testing setting content to an existing XML object converted
to an XML string...\n" );
setXML( xmlObjectConverted, false );
```

JavaScript method: XML.setNodeValue()

This method adds or updates the value of the current element. It requires you use the other XML get methods to navigate through an XML document.

Syntax

`XML.setNodeValue( String );`

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
String	String	This argument specifies the text string you want the script to add as the XML element value. The string argument must contain characters valid for XML (for example, the string cannot include the characters < or > except as XML entities such as < and >).

Return values

An XML object or null.

The method returns an XML object containing the new value of the current XML element or returns null if the method cannot set a value for the element.

Example

This example sets the value of the elements in an XML document.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test03.xml" containing "<document><parent1></parent1><parent2></parent2></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test03.xml", true );
var TargetNode;
var TargetValue;
var findElement;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */

function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
  {
   node = childNode;
  }
 }
 return node;
}
```

```
/* Create a function to print the target element and search results
*/
function setElementValue( searchResult, targetNode, newValue )
{
 print( "The target element we are looking for is: " + targetNode );
 if ( searchResult != null )
 {
  var elementName = searchResult.getNodeName();
  print( "Found element " + elementName + " in: \n" + searchResult +
"\n" );
  searchResult.setNodeValue( newValue );
  print( "The new value of " + targetNode + " is: " + searchResult
);
  return searchResult
 }
 else
 {
  print( "Cannot find " + targetNode );
  return null
 }
}

TargetNode = "parent1";
TargetValue = "1234";
findElement = findTargetElement( XMLFile, TargetNode );
setElementValue( findElement, TargetNode, TargetValue );
```

JavaScript method: XML.setText()

This method adds or updates the value of the current element. You must use the other XML get methods to navigate through an XML document. This method is a synonym for setNodeValue();.

Syntax

```
XML.setText( String );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
String	String	This argument specifies the text string you want the script to add as the XML element value. This argument must contain characters valid for XML (for example, the string cannot include the characters < or > except as XML entities such as < and >).

Return values

An XML object or null.

The method returns an XML object containing the new value of the current XML element or returns null if the method cannot set a value for the element.

Example

This example sets the value of the elements in an XML document.
This example requires the following sample data:

- A local XML file (for example, a file called "C:\test03.xml" containing "<document><parent1></parent1><parent2></parent2></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test03.xml", true );
var TargetNode;
var TargetValue;
var findElement;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */

function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
     return null;
    }
    childNode = node.getNextSiblingElement();
   }
   node = childNode;
  }
  else
```

```
    {
     node = childNode;
    }
   }
   return node;
  }

  /* Create a function to print the target element and search results
  */
  function setElementValue( searchResult, targetNode, newValue )
  {
   print( "The target element we are looking for is: " + targetNode );
   if ( searchResult != null )
   {
    var elementName = searchResult.getNodeName();
    print( "Found element " + elementName + " in: \n" + searchResult +
  "\n" );
    searchResult.setText( newValue );
    print( "The new value of " + targetNode + " is: " + searchResult
  );
    return searchResult
   }
   else
   {
    print( "Cannot find " + targetNode );
    return null
   }
  }

  TargetNode = "parent1";
  TargetValue = "1234";
  findElement = findTargetElement( XMLFile, TargetNode );
  setElementValue( findElement, TargetNode, TargetValue );
```

JavaScript method: XML.setValue()

This method adds or updates the value of the current element. You must use the other XML get methods to navigate through an XML document. This method is a synonym for setNodeValue();.

Syntax

```
XML.setValue( String );
```

Arguments

The following arguments are valid for this method:

Argument	Data type	Description
String	String	This argument specifies the text string you want the script to add as

		the XML element value. The string argument must contain characters valid for XML (for example, the string cannot include the characters < or > except as XML entities such as < and >).

Return values

An XML object or null.

The method returns an XML object containing the new value of the current XML element or returns null if the method cannot set a value for the element.

Example

This example sets the value of the elements in an XML document.

This example requires the following sample data:

- A local XML file (for example, a file called "C:\test03.xml" containing "<document><parent1></parent1><parent2></parent2></document>")

```
/* Create variables to store XML objects and strings.
*  Use XMLObject to store a valid XML object such as an XML file.
*  Use sourceXMLString to store a valid XML string.
*  The script sets the content of XMLString to sourceXMLString.
*  The script assigns values to the empty variables later on */
var XMLFile = new XML();
XMLFile = XMLFile.setContent( "C:\\test03.xml", true );
var TargetNode;
var TargetValue;
var findElement;

/* Create a function that searches an XML object (node) for a
particular
*  element (targetElem) and returns an XML object containing the
element.
*  This function assumes that there is only one instance of the
target
*  element in the source data. If it finds more than one element in
the
*  data it only returns the first instance of the element. */

function findTargetElement( node, targetElem )
{
 var topNodeName = node.getNodeName();
 while (node != null && node.getNodeName() != targetElem )
 {
  var childNode = node.getFirstChildElement();
  if (childNode == null)
  {
   childNode = node.getNextSiblingElement();
   while (childNode == null)
   {
    node = node.getParentNode();
    if ( node == null || topNodeName == node.getNodeName() )
    {
```

```
        return null;
       }
       childNode = node.getNextSiblingElement();
      }
      node = childNode;
     }
     else
     {
      node = childNode;
     }
    }
    return node;
   }

   /* Create a function to print the target element and search results
   */
   function setElementValue( searchResult, targetNode, newValue )
   {
    print( "The target element we are looking for is: " + targetNode );
    if ( searchResult != null )
    {
     var elementName = searchResult.getNodeName();
     print( "Found element " + elementName + " in: \n" + searchResult +
   "\n" );
     searchResult.setValue( newValue );
     print( "The new value of " + targetNode + " is: " + searchResult
   );
     return searchResult
    }
    else
    {
     print( "Cannot find " + targetNode );
     return null
    }
   }

   TargetNode = "parent1";
   TargetValue = "1234";
   findElement = findTargetElement( XMLFile, TargetNode );
   setElementValue( findElement, TargetNode, TargetValue );
```

JavaScript method: XML.toXMLString()

This method converts an XML object into a valid XML string. You must use the other XML get methods to navigate through an XML document.

Syntax

```
XML.toXMLString();
```

Arguments

There are no arguments for this method.

Return values

A string or null.

The method returns a string containing valid XML or returns null if the current node is not an XML object.

Example

This example converts an XML object into a valid XML document.

This example requires the following sample data:

- An existing XML object (for example, the list of currently logged on users stored in system.users)

```
var xmlObject = system.users
print( "This is the original XML object:\n" + xmlObject );
var xmlString = xmlObject.toXMLString();
print( "This is the XML object after conversion to a string:\n" +
xmlString );
```

JavaScript method: xmlDoc.insertBefore()

Inserts the specified node before a reference element as a child of the current node.

Syntax

```
var insertedElement = parentElement.insertBefore(newElement,
referenceElement)
```

Arguments

The following arguments are valid for this function:

Name	Data type	Required	Description
insertedElement	XML	Yes	The node being inserted, that is `newElement`.
parentElement	XML	Yes	The parent of the newly inserted node.
newElement	XML	Yes	The node to insert.
referenceElement	XML	Yes	The parent of the newly inserted node.

Usage Notes

1. The `referenceElement` may NOT be null. If `referenceElement` is null, `newElement` the function will fail.
2. The `newElement` must be a newly created element within the same document (created using importNode or createNode) which has not yet been attached to the tree, such as a newly created element for which appendNode has not been called. The result of invoking `insertBefore` using references to nodes already in the same DOM tree as `newElement` is undefined.

Example

```
var xmlDoc = new XML();
var someXML = "<doc> \
                  <one/> \
                  <two/> \
                </doc>";
  // Create an XML document from the literal XML string
  // we pass "false" as the second argument to setContent() to
indicate
  // that the supplied string does NOT represent a path to an XML
file

  xmlDoc.setContent( someXML, false );

  // Now add create and add new element <four> in one step.
  // It will be added at the end, i.e. after <two>

  var nodeFour = xmlDoc.addElement( "four" );

  // Now insert a new element <three> in front of <four>
  // We do this in two steps:
  // a) create a new element node called <three>
  // b) insert the new element node before the <four> element
  // Note: Passing "1" as the first argument to createNode means
create an element

  var nodeThree = xmlDoc.createNode( 1, "three" );

  // The newly created node has not yet been given a position in the
tree.
  // Just as in the case of a node created with "importNode", we
must
  // attach the new node somewhere. We can do that with appendNode
or insertBefore
  // To get the new node in between two sibling nodes (as in this
example)
  // we must use insertBefore. appendChild can only be used on a
parent node,
  // and if we were to append to the parent element <doc>, it would
end up following <four>

  xmlDoc.insertBefore( nodeThree, nodeFour );

  print( xmlDoc );
```

JavaScript object: XMLDate

The following JavaScript object is unique to HP Service Manager.

With no arguments, the XMLDate() constructor creates an empty XMLDate object.

With a DateObject argument, the XMLDate() constructor converts a JavaScript date object to an XMLDate object.

With an ISO8601DateTimeOrDurationString argument, the XMLDate() constructor converts an XML schema date object to an XMLDate object.

With a DateTimeDatum argument, the XMLDate() constructor converts a Service Manager date object to an XMLDate object.

This object's methods use the Service Manager-defined global return codes.

Constructor

```
new XMLDate();
new XMLDate( DateObject );
new XMLDate( ISO8601DateTimeOrDurationString );
new XMLDate( DateTimeDatum );
```

Arguments

The following arguments are valid for this object:

Argument	Data type	Description
DateObject	JavaScript date object	Passes a JavaScript date object to the XMLDate constructor.
ISO8601DateTimeOrDurationString	XML schema date	Passes an XML schema date, time, datetime, or duration string to the XMLDate constructor.
DateTimeDatum	Service Manager Datum object	Passes a Service Manager datum object formatted as a TIME type to the XMLDate constructor.

Properties

None

Methods

The following methods are valid for this object:

Defined method	Description
getISODateTimeString	This method passes a JavaScript Date object to the constructor.
getISODate	This method returns the indicated portions of the XMLDate value.
getISOTime	This method returns the indicated portions of the XMLDate value.
getISOYear	This method returns the indicated portions of the XMLDate value.
getISOMonth	This method returns the indicated portions of the XMLDate value.

getISODay	This method returns the indicated portions of the XMLDate value.
getDatum	This method returns an SCDatum object.
getSCDateTimeString	This method returns a datetime string in Service Manager format.
getGMTSCDateTimeString	This method returns the Greenwich Mean Time (GMT) datetime in Service Manager format.
toSCDuration	This method converts an ISO duration string to a Service Manager duration.
addDuration (iso8601durationstring)	This method adds the indicated duration to the contained value in the XMLDate object.
JSDate	This method returns a JavaScript date object.
getDate	This method returns a JavaScript date object (same as the JSDate method).

Example

This example does the following:

- Creates an SCFile object from the cm3r table
- Creates an XMLDate object
- Sets the value of two fields to the XMLdate

This example requires the following sample data:

- A valid duration

```
/* Instantiate a new SCFile object with records of the cm3r table */
var changeRequest = new SCFile("cm3r");

/* Set the Service Manager fields date.entered and planned.start to
today's date.
*  The dots in Service Manager field names must be converted to
underscores */
var theXMLDate = new XMLDate( new Date() );
var todaysDate = theXMLDate.getDatum();
changeRequest.header.date_entered = todaysDate;
changeRequest.header.planned_start = todaysDate;

/* Increment the value of the planned.end field to be a week later.
*  P is the addDuration parameter for duration
*  7D is the addDuration parameter for 7 days
*  See the addDuration method for additional information */
theXMLDate.addDuration('P7D');
var nextWeeksDate = theXMLDate.getDatum();
changeRequest.header.planned_end = nextWeeksDate;
```

```
changeRequest.doInsert();
```

JavaScript method: XMLDate.addDuration()

This method returns an XMLDate object containing the new date/time after adding the duration from the duration argument.

Syntax

```
XMLDate.addDuration( duration );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
duration	ISO 8601 duration string	Yes	This argument contains the ISO 8601 duration string or variable value you want the script to add to an XMLDate object.

Return values

An XMLDate object or null.
The method returns an XMLDate object or null if conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing today's date
- Adds 1 year, 2 months, 3 days, and 4 hours to the XMLDate object then displays the result

```
print( "Creating XMLDate object..." );
var d = new XMLDate( new Date() );
print( "The value of the new XMLDate object is: " + d );
print( "Adding 1 year, 2 months, 3 days, and 4 hours to date..." );
var dur = d.addDuration( "P1Y2M3DT4H" );
print( "The value of the new XMLDate object is: " + dur );
```

JavaScript method: XMLDate.getDate()

This method returns a JavaScript date/time object.

Syntax

```
XMLDate.getDate();
```

Arguments

There are no arguments for this method.

Return values

A string or null.
The method returns a JavaScript date/time object or null if conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing today's date
- Converts the XMLDate object to JavaScript date/time object then displays the result

```
print( "Creating XMLDate object..." );
var d = new XMLDate( new Date() );
print( "The value of the new XMLDate object is: " + d );
print( "Converting the XMLDate to a JavaScript object..." );
var js = d.getDate();
print( "The JavaScript date/time object is: " + js );
```

JavaScript method: XMLDate.getDatum()

This method returns an HP Service Manager-formatted date/time Datum object.

Syntax

```
XMLDate.getDatum();
```

Arguments

There are no arguments for this method.

Return values

A string or null.
The method returns a Service Manager-formatted date/time Datum object or null if the conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing today's date
- Converts the XMLDate object to Service Manager-formatted date/time Datum object then displays the result

```
print( "Creating XMLDate object..." );
var d = new XMLDate( new Date() );
print( "The value of the new XMLDate object is: " + d );
print( "Converting the XMLDate to a Service Manager Datum..." );
var sm = d.getDatum();
print( "The Service Manager Datum object is: " + sm );
```

JavaScript method: XMLDate.getGMTSCDateTimeString()

This method returns HP Service Manager-formatted date/time string in Greenwich Mean Time (GMT).

Syntax

```
XMLDate.getSCDateTimeString();
```

Arguments

There are no arguments for this method.

Return values

A string or null.
The method returns a Service Manager-formatted date/time string or null if the conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing today's date
- Converts the XMLDate object to Service Manager-formatted date/time string then displays the result

```
print( "Creating XMLDate object..." );
var d = new XMLDate( new Date() );
print( "The value of the new XMLDate object is: " + d );
print( "Converting the XMLDate to a Service Manager date/time
string..." );
var sc = d.getGMTSCDateTimeString();
print( "The Service Manager GMT date/time string is: " + sc );
```

JavaScript method: XMLDate.getISODate()

This method returns a string containing the date portion of date/time object in ISO format.

Syntax

```
XMLDate.getISODate();
```

Arguments

There are no arguments for this method.

Return values

A string or null.
The method returns a string containing the ISO date of a date/time object or null if conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing today's date
- Converts the XMLDate object to an ISO date then displays the result

```
print( "Creating XMLDate object..." );
var d = new XMLDate( new Date() );
print( "The value of the new XMLDate object is: " + d );
print( "Converting the XMLDate to ISO format..." );
var ISO = d.getISODate();
print( "The ISO date is: " + ISO );
```

JavaScript method: XMLDate.getISODateTimeString()

This method returns a string containing the date and time portion of date/time object in ISO format.

Syntax

```
XMLDate.getISODateTimeString();
```

Arguments

There are no arguments for this method.

Return values

A string or null.
The method returns a string containing the ISO date and time of a date/time object or null if the conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing today's date
- Converts the XMLDate object to an ISO date and time then displays the result

```
print( "Creating XMLDate object..." );
var d = new XMLDate( new Date() );
print( "The value of the new XMLDate object is: " + d );
print( "Converting the XMLDate to ISO format..." );
var ISO = d.getISODateTimeString();
print( "The ISO date and time is: " + ISO );
```

JavaScript method: XMLDate.getISODay()

This method returns a string containing the day portion of date/time object in ISO format.

Syntax

```
XMLDate.getISODay();
```

Arguments

There are no arguments for this method.

Return values

A string or null.
The method returns a string containing the ISO day of a date/time object or null if the conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing today's date
- Converts the XMLDate object to an ISO day then displays the result

```
print( "Creating XMLDate object..." );
var d = new XMLDate( new Date() );
print( "The value of the new XMLDate object is: " + d );
print( "Converting the XMLDate to ISO format..." );
var ISO = d.getISODay();
print( "The ISO day is: " + ISO );
```

JavaScript method: XMLDate.getISOMonth()

This method returns a string containing the month portion of date/time object in ISO format.

Syntax

```
XMLDate.getISOMonth();
```

Arguments

There are no arguments for this method.

Return values

A string or null.
The method returns a string containing the ISO month of a date/time object or null if the conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing today's date
- Converts the XMLDate object to an ISO month then displays the result

```
print( "Creating XMLDate object..." );
var d = new XMLDate( new Date() );
print( "The value of the new XMLDate object is: " + d );
print( "Converting the XMLDate to ISO format..." );
var ISO = d.getISOMonth();
print( "The ISO month is: " + ISO );
```

JavaScript method: XMLDate.getISOTime()

This method returns a string containing the time portion of date/time object in ISO format.

Syntax

```
XMLDate.getISOTime();
```

Arguments

There are no arguments for this method.

Return values

A string or null.
The method returns a string containing the ISO time of a date/time object or null if the conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing today's date
- Converts the XMLDate object to an ISO time then displays the result

```
print( "Creating XMLDate object..." );
var d = new XMLDate( new Date() );
```

```
print( "The value of the new XMLDate object is: " + d );
print( "Converting the XMLDate to ISO format..." );
var ISO = d.getISOTime();
print( "The ISO time is: " + ISO );
```

JavaScript method: XMLDate.getISOYear()

This method returns a string containing the year portion of date/time object in ISO format.

Syntax

```
XMLDate.getISOYear();
```

Arguments

There are no arguments for this method.

Return values

A string or null.
The method returns a string containing the ISO year of a date/time object or null if the conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing today's date
- Converts the XMLDate object to an ISO year then displays the result

```
print( "Creating XMLDate object..." );
var d = new XMLDate( new Date() );
print( "The value of the new XMLDate object is: " + d );
print( "Converting the XMLDate to ISO format..." );
var ISO = d.getISOYear();
print( "The ISO year is: " + ISO );
```

JavaScript method: XMLDate.getSCDateTimeString()

This method returns an HP Service Manager-formatted date/time string.

Syntax

```
XMLDate.getSCDateTimeString();
```

Arguments

There are no arguments for this method.

Return values

A string or null.
The method returns a Service Manager-formatted date/time string or null if the conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing today's date
- Converts the XMLDate object to Service Manager-formatted date/time string then displays the result

```
print( "Creating XMLDate object..." );
var d = new XMLDate( new Date() );
print( "The value of the new XMLDate object is: " + d );
print( "Converting the XMLDate to a Service Manager date/time
string..." );
var sc = d.getSCDateTimeString();
print( "The Service Manager date/time string is: " + sc );
```

JavaScript method: XMLDate.JSDate()

This method returns a JavaScript date/time object. This method is an alias of the getDate() method.

Syntax

```
XMLDate.JSDate();
```

Arguments

There are no arguments for this method.

Return values

A string or null.
The method returns a JavaScript date/time object or null if the conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing today's date
- Converts the XMLDate object to a JavaScript date/time object then displays the result

```
print( "Creating XMLDate object..." );
var d = new XMLDate( new Date() );
print( "The value of the new XMLDate object is: " + d );
print( "Converting the XMLDate to a JavaScript object..." );
var js = d.JSDate();
print( "The JavaScript date/time object is: " + js );
```

JavaScript method: XMLDate.toSCDuration()

This method converts an ISO 8601 duration string to an HP Service Manager duration string.

Syntax

```
XMLDate.toSCDuration( duration );
```

Arguments

The following arguments are valid for this function:

Argument	Data type	Required	Description
duration	ISO 8601 duration string	Yes	This argument contains the ISO 8601 duration string or variable value you want the script to convert to a Service Manager duration string.

Return values

A string or null.

The method returns a Service Manager duration string or null if the conversion fails.

Example

This example does the following:

- Creates a new XMLDate object containing a hard-coded ISO date
- Converts the ISO date/time to a Service Manager duration string

```
print( "Creating XMLDate object..." );
var d = new XMLDate( "P01DT12H01M01S" );
print( "The value of the new XMLDate object is: " + d );
print( "Converting " + d.toString() + " to a SC duration..." );
var smDur = d.toSCDuration();
print( "The Service Manager duration is: " + smDur );
```

List: JavaScript functions

The following table lists some of the JavaScript functions available in Service Manager. These functions provide a method of performing certain commands that will return a value when executed.

Refer to "List: RAD functions" on page 79 for more information on using RAD functions to perform these commands instead of JavaScript.

JavaScript function	Description
add_ graphnodes	Returns XML to the Run-Time Environment (RTE) for the expand action on a node in the graph diagram.
getBinary	Returns the binary representation of a field.
get_graph_ action	Returns the activated action on a node in the graph diagram.
get_graph_id	Returns the value of the graph ID for the expand action on a node in the graph diagram.
get_graph_ node_id	Returns the value of the node ID for the expand action on a node in the graph diagram.
get_graph_ target	Returns the target for an activated action on a node in the graph diagram.

parse.evaluate ()	Returns the value of a given expression.

JavaScript function: add_graphnodes

This function returns an XML file to the Run-Time Environment (RTE) for the expand action on a node in the graph diagram.

Function

add_graphnodes()

Example

```
system.functions.add.graphnodes(String strXML)
```

JavaScript function: getBinary

Description

A JavaScript function that returns the binary representation of a field.

Function

getBinary()

Example

```
var bb = new SCFile("bbtosysattachments");
var rc = bb.doSelect("true");
var attachmentObj= new Attachment();
attachmentObj.value = bb.getBinary("att_attachment")
attachmentObj.name = bb.att_filename;
attachmentObj.type = "excel";
```

JavaScript function: get_graph_action

Description

A JavaScript function that returns the activated action on a node in the graph diagram.

Function

get_graph_action()

Example

```
system.functions.get_graph_action()
```

JavaScript function: get_graph_id

Description

A JavaScript function that returns the value of the graph ID for the expand action on a node in the graph diagram.

Function

get_graph_id.htm()

Example

```
system.functions.get_graph_id.htm()
```

JavaScript function: get_graph_node_id

Description

A JavaScript function that returns the value of the node ID for the expand action on a node in the graph diagram.

Function

get_graph_node_id()

Example

```
system.functions.get_graph_node_id()
```

JavaScript function: get_graph_target

Description

A JavaScript function that returns the target for an activated action on a node in the graph diagram.

Function

get_graph_target

Example

```
system.functions.get_graph_target()
```

List: JavaScript functions in the ScriptLibrary

The following table lists some of the JavaScript functions available in the ScriptLibrary of Service Manager. These functions are part of system JavaScript files and should not be directly modified.

JavaScript function	Implemented in version
addToGroup	6.2
checkAllowedOption	6.2
isInGroup	6.2
removeFromGroup	6.2
returnGroupMembers	6.2
returnMyGroups	6.2

JavaScript function: addToGroup

This function adds one or more configuration items (CIs) to a Configuration Management list group and updates the list group record if successful. It is part of a system JavaScript and should not be directly modified.

Syntax

```
lib.ciGrouping.addToGroup( ci, groupName );
```

Arguments

The following arguments are valid for this function:

Name	Data type	Required	Description
ci	Datum object or string	Yes	This argument contains the CI records you want to add. To add an array of CIs, the *ci* argument must be an SCFile or SCDatum object. To add a single CI, you can specify it as a string.
groupName	String	Yes	This argument contains the name of the configuration group to which you want to add CIs.

Return values

A numerical value: 0 or 1.

The function returns 0 if it is unable to add CIs to the list group or if the *groupName* argument specifies a query group. The function returns 1 if the function successfully adds the CIs to the list group.

Example

This example attempts to add CIs to a list group from the following sources:

- The logical.name value from a hard-coded string
- The logical.name value from a configuration group record
- The logical.name values from an array of CI records
- The logical.name values from a record list of one or more CIs

This example requires the following sample data:

- Create a list group containing a single configuration item (for example, a list group called "test01" containing the configuration item "DEFAULT Phone 0001")
- Create a list group containing one or more configuration items (for example, a list group called "test02" containing the configuration item "DEFAULT Phone 0003")
- Create a list group containing one or more configuration items (for example, a list group called "test03" containing the configuration items "HH-000001" and "HH-000002")

```
/* Create variables to store CI group names, strings, and datum
objects.
*  Use targetCIGroup to store the name of CI group you want to add
records to.
*  Use sourceCIGroup to store the name of CI group you want to
```

```
create an array from.
*  Use oneLogicalName to store a string containing a logical.name
you want to add.
*  Use deviceType to store a device type you want to query for.
*  Use startsWith to store a alphabetical value you want to query
for.
*  The variable recordTest stores the output of the findCIGroup
function.
*  The variable arrayTest stores the output of the createArray
function.
*  The variable recordListTest stores the output of the
findRecordList function. */
var targetCIGroup;
var sourceCIGroup;
var oneLogicalName;
var deviceType;
var startsWith;
var recordTest;
var arrayTest;
var recordListTest;

/* Create a function to store a particular CI group in memory. */
function findCIGroup( ciGroup )
{
 print( "Looking for CI Group " + ciGroup + "..." );
 var members = lib.ciGrouping.returnGroupMembers( ciGroup )
 var ciGroupRecord = new SCFile( "cigroup" );
 var queryTest = ciGroupRecord.doSelect( "logical.name=\"" + ciGroup
+ "\"" )
 if ( queryTest == RC_SUCCESS )
 {
  system.vars.$L_file = ciGroupRecord
  print( "Found CI group record for " + ciGroup + "\nThe record
is:\n" + ciGroupRecord );
  print( "The current members of " + ciGroup + " are:\n" + members
);
  return ciGroupRecord
 }
 else
 {
  print( "Cannot find record for " + ciGroup + ". " +  RCtoString(
queryTest ) + "." );
  return null;
 }
}

/* Create function to find a record list based on a device type and
starting query */
function findRecordList( deviceType, query )
{
```

```
 var ciRecordList = new SCFile( "device" );
 if ( query != null )
 {
  print( "Looking for CIs matching type=\"" + deviceType + "\"\&" +
"logical.name#\"" + query + "\"" );
  var queryTest = ciRecordList.doSelect( "type=\"" + deviceType +
"\"\&" + "logical.name#\"" + query + "\"" )
 }
 else
 {
  print( "Looking for CIs matching type=\"" + deviceType + "\"" );
  var queryTest = ciRecordList.doSelect( "type=\"" + deviceType +
"\"" )
 }
 if ( queryTest == RC_SUCCESS )
 {
  print( "Found the following CIs:\n" + ciRecordList );
  system.vars.$L_file = ciRecordList
  return ciRecordList;
 }
 else
 {
  if ( query != null )
  {
   print( "Cannot find records matching query: device=\""+
deviceType + "\"&logical.name#\"" + query + "\""
          +  RCtoString( queryTest ) + "." );
   return null;
  }
  else
  {
   print( "Cannot find records matching query: device=\""+
deviceType + "\"" +  RCtoString( queryTest ) + "." );
   return null;
  }
 }
}

/* Create a function to store the logical.names of the record. */
function createArray( record )
{
 var arrayOfLogicalNames = record.group_members;
 if (arrayOfLogicalNames == null )
 {
  print( "Cannot create array from " + record + ". Group not in the
cigroup table." );
  return null
 }
 else
 {
```

```
  print( "Created array from " + record + ".\nThe array is:\n" +
arrayOfLogicalNames );
  return arrayOfLogicalNames
 }
}

/* Create a function to call the addToGroup function and print
results. */
function testAdd( ci, group )
{
 var groupMembers = lib.ciGrouping.returnGroupMembers( group )
 var AddCI = lib.ciGrouping.addToGroup( ci, group );
 var afterAdd = lib.ciGrouping.returnGroupMembers( group );
 print( "Now trying to add configuration item(s) to " + group +
"..." );
 print( "The current members of " + group + " are:\n" + groupMembers
);

 if ( AddCI == 1 )
 {
  print( "Successfully added configuration item(s) to " + group );
  print( "The members of " + group + " are now: \n" + afterAdd );
 }
 else
 {
  print( "Could not add configuration item(s) to " + group );
  print( "The members of " + group + "are : \n" + groupMembers );
 }
}

/* Test adding a CI to a group from a string */
print( "Now testing adding a single CI from a string...\n" );
targetCIGroup = "test01";
oneLogicalName = "DEFAULT Phone 0002";
members = lib.ciGrouping.returnGroupMembers( targetCIGroup )
testAdd( oneLogicalName, targetCIGroup );

/* Test adding a CI to a group from an existing CI record */
print( "Now testing adding a CI member from a CI group record...\n"
);
targetCIGroup = "test01";
sourceCIGroup = "test02";
recordTest = findCIGroup( sourceCIGroup );
if ( recordTest != null )
{
 testAdd( system.vars.$L_file, targetCIGroup );
}

/* Test adding one or more CIs from an array of logical,names */
print( "Now testing adding one or more CIs from an array of logical
```

```
names...\n" );
targetCIGroup = "test01";
sourceCIGroup = "test03";
recordTest = findCIGroup( sourceCIGroup );
if ( recordTest != null )
{
 arrayTest = createArray( recordTest );
 testAdd( arrayTest, targetCIGroup );
}

/* Test adding one or more CIs to a group from a CI record list */
print( "Now testing adding one or more CIs from a CI record
list...\n" );
targetCIGroup = "test01";
deviceType = "furnishings"
startsWith = null;
recordListTest = findRecordList( deviceType, startsWith );
if ( recordListTest != null )
{
 testAdd( system.vars.$L_file, targetCIGroup );
}
```

JavaScript function: checkAllowedOption

Returns the next valid version number of a Configuration Management baseline group if there is only one valid version number available.

Syntax

```
lib.versionControl.checkAllowedOption( oldVersion, maxIncrement );
```

Arguments

The following arguments are valid for this function:

Name	Data type	Required	Description
oldVersion	String	Yes	This argument contains the previous version number.
maxIncrement	Array	Yes	This argument contains an array of numbers representing the decimal value of the maximum number increments possible. For example, if *maxIncrement* equals [0,1,0] then there are ten possible increments permissible from versions 0.0.1 to 0.1.0.

Return values

A valid version number or the string "more".

The function returns a valid version number if there is one and only one valid next version number. The function returns the string "more" if there is more than one valid next version number.

Description

This function is part of a system JavaScript and should not be directly modified.
This function returns the next valid version number of a Configuration Management baseline group if there is one and only one valid version number available. If there is more than one valid next version number then the function returns the string "more".

Example

This example attempts to create the next valid version from two variables.
This example requires the following sample data:

- A variable storing the old version number
- A variable storing the maximum number of increments allowed

```
/* Use oldVersion to store the old version number as a string.
*  Use maxIncrement to store the maximum number of increments as an
array. */
var oldVersion;
var maxIncrement = new Array();

/* Create a function to test the checkAllowedOption function */
function nextVersionTest( versionOld, increment )
{
 var nextVersion = lib.versionControl.checkAllowedOption(
versionOld, increment );
 if ( nextVersion == "more" )
 {
  print( "Error. There is more than one valid next version number
for " + versionOld
               + ".\n");
  return versionOld
 }
 else
 {
  print( "Success. The next valid version number is " + nextVersion
+ ".\n" );
  return nextVersion
 }
}

/* Test a valid version number update */
print( "Testing checkAllowedOption...\n" );
oldVersion = "1.0";
maxIncrement = [0,1];
print( "The current version number is " + oldVersion + "." );
print( "The maximum allowed increment is " + maxIncrement + ".");
nextVersionTest( oldVersion, maxIncrement );

/* Test a valid version number update */
print( "Testing checkAllowedOption...\n" );
oldVersion = "0.9.9";
maxIncrement = [0,0,1];
```

```
print( "The current version number is " + oldVersion + "." );
print( "The maximum allowed increment is " + maxIncrement + ".");
nextVersionTest( oldVersion, maxIncrement );

/* Test an invalid version number update */
print( "Testing error condition...\n" );
oldVersion = "1.0";
maxIncrement = [0,2];
print( "The current version number is " + oldVersion + "." );
print( "The maximum allowed increment is " + maxIncrement + ".");
nextVersionTest( oldVersion, maxIncrement );
```

JavaScript function: checkVersionString

Validates a version string to ensure it adheres to Configuration Management baseline group version number requirements.

Syntax

```
lib.versionControl.checkVersionString( oldVersion, newVersion,
maxIncrement );
```

Arguments

The following arguments are valid for this function:

Name	Data type	Required	Description
oldVersion	String	Yes	This argument contains the previous version number.
newVersion	String	Yes	This argument contains the new version number.
maxIncrement	Array	Yes	This argument contains an array of numbers representing the decimal value of the maximum number increments possible. For example, if *maxIncrement* equals [0,1,0] then there are ten possible increments permissible from versions 0.0.1 to 0.1.0.

Return values

An integer value: 0, 1, -1, -2, -3, or -4.

Return value	Condition
0	The function returns 0 if any version level value in the *newVersion* argument exceeds the value in the *maxIncrement* argument. For example, if *oldVersion* equals "1.0". and *maxIncrement* equals "2", then the function returns a value of 0 if *newVersion* equals "1.3" since going from a value of zero to three exceeds the *maxIncrement* value of 2.

1	The function returns 1 if it successfully validates the proposed change from the *oldVersion* to the *newVersion*.
-1	The function returns -1 if the *newVersion* argument contains greater than five version levels. For example, the function returns -1 if the *newVersion* argument equals "1.0.0.0.0.0" as this is a version with six version levels.
-2	The function returns -2 if a version level value in the *newVersion* argument contains anything other than the digits 0 to 9. For example the function returns -2 if the *newVersion* argument equals "gold" or "1.0a" because the version number contains alphabetical characters.
-3	The function returns -3 if the number of version levels in the *oldVersion* argument do not match the number of version levels in the *newVersion* argument. For example, the function returns -3 if *oldVersion* equals "1.1" and *newVersion* equals "1.1.1" since the *oldVersion* has two version levels and the *newVersion* has three.
-4	The function returns -4 if the value of the *newVersion* argument is less than the value of the *oldVersion* argument. For example, the function returns -4 if *oldVersion* equals "1.1" and *newVersion* equals "1.0" because the *newVersion* value is prior to the *oldVersion* value.

Description

This function is part of a system JavaScript and should not be directly modified.
This function validates a new Configuration Management baseline group version string against an old version string to ensure it adheres to version number requirements.

Example

This example attempts to validate any given version number update from three variables.
This example requires the following sample data:

- A variable storing the old version number
- A variable storing the desired new version number
- A variable storing the maximum number of increments allowed

```
/* Use oldVersion to store the old version number as a string.
*  Use newVersion to store the new version number as a string.
*  Use maxIncrement to store the maximum number of increments as an
array. */
var oldVersion;
var newVersion;
var maxIncrement = new Array();

/* Create a function to test the checkVersionString function */
function versionTest( versionOld, versionNew, increment )
{
 var validate = lib.versionControl.checkVersionString( versionOld,
versionNew, increment );
 print( "The checkVersionString return value is " +  validate );
 if ( validate == 1 )
 {
```

```
  print( "Success. Updating " + versionOld + " to " + versionNew + "
is a valid change.\n");
  return versionNew
 }
 if ( validate == 0 )
 {
  print( "Error. Could not update " + versionOld + " to " +
versionNew + "." );
  print( "The new version " + versionNew + " exceeds the maximum
version increment value of "
               + increment + "." );
  return versionOld
 }
 if ( validate == "-1" )
 {
  print( "Error. Could not update " + versionOld + " to " +
versionNew + ".");
  print( "The new version " + versionNew + " contains more than five
version levels." );
  return versionOld
 }
 if ( validate == "-2" )
 {
  print( "Error. Could not update " + versionOld + " to " +
versionNew + ".");
  print( "The new version " + versionNew + " contains characters
other than 0 to 9." );
  return versionOld
 }
 if ( validate == "-3" )
 {
  print( "Error. Could not update " + versionOld + " to " +
versionNew + ".");
  print( "The new version " + versionNew
               + " contains a different number of version levels than the old
version " + versionOld );
  return versionOld
 }
 if ( validate == "-4" )
 {
  print( "Error. Could not update " + versionOld + " to " +
versionNew + ".");
  print( "The new version " + versionNew + " is an earlier version
than the old version "
               + versionOld );
  return versionOld
 }
 else
 {
  return versionOld
```

```
 }
}

/* Test a valid version number update */
print( "Testing checkVersionString...\n" );
oldVersion = "1.0";
newVersion = "1.1";
maxIncrement = [0,2];
versionTest( oldVersion, newVersion, maxIncrement );

/* Test an invalid version number update */
print( "Testing error condition 0...\n" );
oldVersion = "1.0";
newVersion = "1.9";
maxIncrement = [0,2];
versionTest( oldVersion, newVersion, maxIncrement );

/* Test an invalid version number update */
print( "Testing error condition -1...\n" );
oldVersion = "1.0.0.0.0.0";
newVersion = "1.0.0.0.0.1";
maxIncrement = [0,0,1,1,2];
versionTest( oldVersion, newVersion, maxIncrement );

/* Test an invalid version number update */
print( "Testing error condition -2...\n" );
oldVersion = "1.0.0.0.0";
newVersion = "1.a.0.0.0";
maxIncrement = [0,0,1,1,2];
versionTest( oldVersion, newVersion, maxIncrement );

/* Test an invalid version number update */
print( "Testing error condition -3...\n" );
oldVersion = "1.0.0";
newVersion = "1.1";
maxIncrement = [0,2,2];
versionTest( oldVersion, newVersion, maxIncrement );

/* Test an invalid version number update */
print( "Testing error condition -4...\n" );
oldVersion = "1.2";
newVersion = "1.1";
maxIncrement = [0,2];
versionTest( oldVersion, newVersion, maxIncrement );
```

JavaScript function: isInGroup

Searches for one configuration item (CI) in a Configuration Management list or query group.

Syntax

```
lib.ciGrouping.isInGroup( ci, groupName );
```

Arguments

The following arguments are valid for this function:

Name	Data type	Required	Description
ci	String	Yes	This argument contains the CI record you want to search for in the group.
groupName	String	Yes	This argument contains the name of the configuration group where you want to search for the CI.

Return values

A Boolean value: true or false.

The function returns true if the function successfully finds the CI in the list or query group. The function returns false if it is unable to find the CI in the list or query group.

Description

This function is part of a system JavaScript and should not be directly modified.

This function searches for a CI in list or query groups and returns a Boolean value of true if successful.

Example

This example attempts to search for a CI in the following groups:

- A list group
- A query group

This example requires the following sample data:

- Create a list group containing one or more configuration items (for example, a list group called "test01" containing the configuration item "DEFAULT Phone 0001")
- Create a query group containing one or more configuration items (for example, a query group called "test04" containing the configuration items where type equals "telecom")

```
/* Create variables to store CI group name and the results of
returnGroupMembers */
var targetCIGroup;
var targetCI;
var members;

function findInGroup( ci, group )
{
 inGroupTest = lib.ciGrouping.isInGroup( ci, group );
 if ( inGroupTest == true )
 {
  print( "Success. Found " + ci + " in " + group );
 }
 else
 {
  print( "Error. Could not find " + ci + " in " + group );
 }
```

```
}

/* Test returnGroupMembers on a list group. */
print( "Now testing isInGroup on list group...\n" );
targetCIGroup = "test01";
targetCI = "DEFAULT Phone 0001";
members = lib.ciGrouping.returnGroupMembers( targetCIGroup )
print( "Looking for " + targetCI + " in " + targetCIGroup + "..." );
findInGroup( targetCI, targetCIGroup );
print( "The current members of " + targetCIGroup + " are:\n" +
members );

/* Test returnGroupMembers on a query group. */
print( "Now testing isInGroup on query group...\n" );
targetCIGroup = "test04";
targetCI = "DEFAULT Phone 0006";
members = lib.ciGrouping.returnGroupMembers( targetCIGroup )
print( "Looking for " + targetCI + " in " + targetCIGroup + "..." );
findInGroup( targetCI, targetCIGroup );
print( "The current members of " + targetCIGroup + " are:\n" +
members );
```

JavaScript function: removeFromGroup

Removes one or more configuration items (CI) from a Configuration Management list group.

Syntax

```
lib.ciGrouping.removeFromGroup( ci, groupName );
```

Arguments

The following arguments are valid for this function:

Name	Data type	Required	Description
ci	Datum object or string	Yes	This argument contains the CI records you want to remove. To remove an array of CIs, the *ci* argument must be an SCFile or SCDatum object. To remove a single CI, you can specify it as a string.
groupName	String	Yes	This argument contains the name of the configuration group from which you want to remove CIs.

Return values

A numerical value: 0, 1, or -1

The function returns 0 if it is unable to remove CIs from the list group or if the *groupName* argument specifies a query group. The function returns 1 if the function successfully removes CIs from the list group. The function returns -1 if it cannot remove CIs from the list group because the action would result in an empty configuration list group.

Description

This function is part of a system JavaScript and should not be directly modified.
This function removes one or more CIs from a configuration list group and updates the list group record if successful.

Example

This example attempts to remove the following CIs from a list group:

- A string containing a single CI record
- A configuration group record containing one or more CIs
- A Datum object containing an array of CI records

This example requires the following sample data:

- Create a list group containing a single configuration item (for example, a list group called "test01" containing the configuration item "DEFAULT Phone 0001")
- Create a list group containing one or more configuration items (for example, a list group called "test02" containing the configuration item "DEFAULT Phone 0003")
- Create a list group containing one or more configuration items (for example, a list group called "test03" containing the configuration items "HH-000001" and "HH-000002")

```
/* Create variables to store CI group names, strings, and datum
objects.
*  Use targetCIGroup to store the name of CI group you want to
remove records from.
*  Use sourceCIGroup to store the name of CI group you want to
create an array from.
*  Use oneLogicalName to store a string containing a logical.name
you want to remove.
*  Use deviceType to store a device type you want to query for.
*  Use startsWith to store a alphabetical value you want to query
for.
*  The variable recordTest stores the output of the findCIGroup
function.
*  The variable arrayTest stores the output of the createArray
function.
*  The variable recordListTest stores the output of the
findRecordList function. */
var targetCIGroup;
var sourceCIGroup;
var oneLogicalName;
var deviceType;
var startsWith;
var recordTest;
var arrayTest;
var recordListTest;

/* Create a function to store a particular CI group in memory. */
function findCIGroup( ciGroup )
{
 print( "Looking for CI Group " + ciGroup + "..." );
 var members = lib.ciGrouping.returnGroupMembers( ciGroup )
```

```
 var ciGroupRecord = new SCFile( "cigroup" );
 var queryTest = ciGroupRecord.doSelect( "logical.name=\"" + ciGroup
+ "\"" )
 if ( queryTest == RC_SUCCESS )
 {
  system.vars.$L_file = ciGroupRecord
  print( "Found CI group record for " + ciGroup + "\nThe record
is:\n" + ciGroupRecord );
  print( "The current members of " + ciGroup + " are:\n" + members
);
  return ciGroupRecord
 }
 else
 {
  print( "Cannot find record for " + ciGroup + ". " +  RCtoString(
queryTest ) + "." );
  return null;
 }
}

/* Create function to find a record list based on a device type and
starting query */
function findRecordList( deviceType, query )
{
 var ciRecordList = new SCFile( "device" );
 if ( query != null )
 {
  print( "Looking for CIs matching type=\"" + deviceType + "\"\&" +
"logical.name#\"" + query + "\"" );
  var queryTest = ciRecordList.doSelect( "type=\"" + deviceType +
"\"\&" + "logical.name#\"" + query + "\"" )
 }
 else
 {
  print( "Looking for CIs matching type=\"" + deviceType + "\"" );
  var queryTest = ciRecordList.doSelect( "type=\"" + deviceType +
"\"" )
 }
 if ( queryTest == RC_SUCCESS )
 {
  print( "Found the following CIs:\n" + ciRecordList );
  system.vars.$L_file = ciRecordList
  return ciRecordList;
 }
 else
 {
  if ( query != null )
  {
   print( "Cannot find records matching query: device=\""+
deviceType + "\"&logical.name#\"" + query + "\""
```

```
          +  RCtoString( queryTest ) + "." );
   return null;
  }
  else
  {
   print( "Cannot find records matching query: device=\""+
deviceType + "\"" +  RCtoString( queryTest ) + "." );
   return null;
  }
 }
}

/* Create a function to store the logical.names of the record. */
function createArray( record )
{
 var arrayOfLogicalNames = record.group_members;
 if (arrayOfLogicalNames == null )
 {
  print( "Cannot create array from " + record + ". Group not in the
cigroup table." );
  return null
 }
 else
 {
  print( "Created array from " + record + ".\nThe array is:\n" +
arrayOfLogicalNames );
  return arrayOfLogicalNames
 }
}

/* Create a function to call the addToGroup function and print
results. */
function testRemove( ci, group )
{
 var groupMembers = lib.ciGrouping.returnGroupMembers( group )
 var removeCI = lib.ciGrouping.removeFromGroup( ci, group );
 var afterRemove = lib.ciGrouping.returnGroupMembers( group );
 print( "Now trying to remove configuration item(s) from " + group +
"..." );
 print( "The current members of " + group + " are:\n" + groupMembers
);

 if ( removeCI == 1 )
 {
  print( "Successfully removed configuration item(s) to " + group );
  print( "The members of " + group + " are now: \n" + afterRemove );
 }
 else
 {
  if ( removeCI == "-1" )
```

```
  {
   print( "Could not remove configuration item from " + group + " as
"
         + ci + " is the last member of the group." );
   print( "The members of " + group + "are : \n" + groupMembers );
  }
  else
  {
   print( "Could not remove configuration item from " + group +
".\n"
         + ci + " or " + group + " does not exist.");
  }
 }
}

/* Test removing a CI to a group from a string */
print( "Now testing removing a single CI from a string...\n" );
targetCIGroup = "test01";
oneLogicalName = "DEFAULT Phone 0002";
members = lib.ciGrouping.returnGroupMembers( targetCIGroup )
testRemove( oneLogicalName, targetCIGroup );

/* Test removing a CI to a group from an existing CI record */
print( "Now testing removing a CI member from a CI group
record...\n" );
targetCIGroup = "test01";
sourceCIGroup = "test02";
recordTest = findCIGroup( sourceCIGroup );
if ( recordTest != null )
{
 testRemove( system.vars.$L_file, targetCIGroup );
}

/* Test removing one or more CIs from an array of logical,names */
print( "Now testing removing one or more CIs from an array of
logical names...\n" );
targetCIGroup = "test01";
sourceCIGroup = "test03";
recordTest = findCIGroup( sourceCIGroup );
if ( recordTest != null )
{
 arrayTest = createArray( recordTest );
 testRemove( arrayTest, targetCIGroup );
}

/* Test removing one or more CIs to a group from a CI record list */
print( "Now testing removing one or more CIs from a CI record
list...\n" );
targetCIGroup = "test01";
deviceType = "furnishings"
```

```
startsWith = null;
recordListTest = findRecordList( deviceType, startsWith );
if ( recordListTest != null )
{
 testRemove( system.vars.$L_file, targetCIGroup );
}
```

JavaScript function: returnGroupMembers

Returns an array containing the group members of a Configuration Management group.

Syntax

```
lib.ciGrouping.returnGroupMembers( groupName );
```

Arguments

The following arguments are valid for this function:

Name	Data type	Required	Description
groupName	String	Yes	This argument contains the name of the configuration group whose members you want to return as an array.

Return values

An array or null.

The function returns an array containing the logical.name values of the group members if there are group members. The function returns null if it cannot find a matching group name or if the group does not have any members.

Description

This function is part of a system JavaScript and should not be directly modified.

This function returns the logical.name values of the members of a Configuration Management group.

- If the group is a list type, the function returns the array from the group configuration item record.
- If the group is a query type, the function returns an array of all the CIs it finds.

Example

This example returns the members of the group listed in the variable.

This example requires the following sample data:

- Create a list group containing one or more configuration items (for example, a list group called "test01" containing the configuration item "DEFAULT Phone 0001")
- Create a query group containing one or more configuration items (for example, a list group called "test04" containing the configuration items where type equals "telecom")

```
/* Create variables to store CI group name and the results of
returnGroupMembers */
var targetCIGroup;
var members;
```

```
/* Test returnGroupMembers on a list group */
print( "Now testing returnGroupMembers on a list group...\n" );
targetCIGroup = "test01";
members = lib.ciGrouping.returnGroupMembers( targetCIGroup );
print( "The current members of " + targetCIGroup + " are:\n" +
members );

/* Test returnGroupMembers on a query group */
print( "Now testing returnGroupMembers on a query group...\n" );
targetCIGroup = "test04";
members = lib.ciGrouping.returnGroupMembers( targetCIGroup );
print( "The current members of " + targetCIGroup + " are:\n" +
members );
```

JavaScript function: returnMyGroups

Returns an array containing all the groups to which a configuration item (CI) belongs.

Syntax

```
lib.ciGrouping.returnMyGroups( ci, listType );
```

Arguments

The following arguments are valid for this function:

Name	Data type	Required	Description
ci	String	Yes	This argument contains the CI record you want to search for in your list and query groups.
listType	String	No	This argument contains a string describing which types of configuration groups you want the function to query. The string can have one of four values: • list – Add this argument to only search list groups. • query – Add this argument to only search query groups. • both – Add this argument to search both list and query groups. • null – If there is a null entry for this argument, the function searches both list and query groups.

Return values

An array of group names.

The function returns an array containing the logical.name values of the list and query groups in which the CI is a member. The function returns an empty array if it cannot find the CI in any list or query group.

Description

This function is part of a system JavaScript and should not be directly modified.
This function returns an array containing the logical.name values of the groups in which the CI is a member. The function queries all group records and does the following:

- If the group is a list type, the function indexes the group.members and if the CI exists in the list group, adds it to the groupArray.
- If the group is a query type, the function uses the RTE function to determine whether the values in the CI record satisfy the search criteria defined by the query.

Example

This example returns the groups containing the CI listed in a variable.
This example requires the following sample data:

- Create a list group containing a single configuration item (for example, a list group called "test01" containing the configuration item "DEFAULT Phone 0001")
- Create a list group containing one or more configuration items (for example, a list group called "test02" containing the configuration item "DEFAULT Phone 0003")
- Create a list group containing one or more configuration items (for example, a list group called "test03" containing the configuration items "HH-000001" and "HH-000002")
- Create a query group containing one or more configuration items (for example, a query group called "test04" containing the configuration items where type equals "telecom")

```
/* Create variables to store CI group name and the results of
returnMyGroups */
var targetCIGroup;
var listTypes;
var groups;

/* Search for a CI in a list group only */
print( "Now testing search for configuration items in list groups
only...\n" );
targetCI = "DEFAULT Phone 0001";
listTypes = "list";
groups = lib.ciGrouping.returnMyGroups( targetCI, listTypes );
if ( groups.length >= 1 )
{
 print( "The list groups containing " + targetCI + " are:\n" +
groups );
}
else
{
 print( "There are no list groups containing " + targetCI );
}

/* Search for a CI in a query group only */
print( "Now testing search for configuration items in query groups
only...\n" );
targetCI = "DEFAULT Phone 0001";
listTypes = "query";
groups = lib.ciGrouping.returnMyGroups( targetCI, listTypes );
```

```
if ( groups.length >= 1 )
{
 print( "The query groups containing " + targetCI + " are:\n" +
groups );
}
else
{
 print( "There are no query groups containing " + targetCI );
}

/* Search for a CI in both list and query groups */
print( "Now testing search for configuration items in both list and
query groups...\n" );
targetCI = "DEFAULT Phone 0001";
listTypes = "both";
groups = lib.ciGrouping.returnMyGroups( targetCI, listTypes );
if ( groups.length >= 1 )
{
 print( "The groups containing " + targetCI + " are:\n" + groups );
}
else
{
 print( "There are no groups containing " + targetCI );
}
```

List: JavaScript examples

The following table lists the JavaScript examples available for review. These examples include sample JavaScript code for the Service Manager-defined JavaScript objects, methods, and properties, as well as code samples that call the system JavaScript functions provided in the Script Library. These JavaScript are intended to work with an out-of-box Service Manager system and sample data.

JavaScript example	Service Manager object, method, or property demonstrated	System functions called
Example: Getting a list of logged on users	• system.users (global property) • print(); (global method) • getName(); (method of the XML object) • getValue(); (method of the XML object)	None
Example: Iterating through a complex XML document	• XML(); (XML object)	None

	• setContent(); (method of the XML object) • getDocumentElement(); (method of the XML object) • getQualifiedName(); (method of the XML object) • getFirstAttribute(); (method of the XML object) • getFirstChildElement(); (method of the XML object) • getNextSiblingElement(); (method of the XML object)	
Example: Query a record using the SCFile object	• SCFile(); (SCFile object) • doSelect(); (method of the SCFile object) • RCtoString(); (method of the SCFile object) • getFirstChildElement(); (method of the XML object) • getName(); (method of the XML object) • getValue(); (method of the XML object)	None
Example: Testing a file name	• SCFile(); (SCFile object) • doSelect(); (method of the SCFile object) • print(); (global method)	None
Example: Updating a field with the current date and time	• SCFile(); (SCFile object) • doSelect(); (method of the SCFile object) • print(); (global method) • system.functions (global property) • Date(); (Date object) • SCDatum(); (SCDatum object) • getType(); (method of the SCDatum object) • doUpdate(); (method of the SCDatum object)	None

	• RCtoString(); (method of the SCDatum object)	

Example: Getting a list of logged-on users

This JavaScript example accesses the built in system.users object to get an XML list of all the currently logged-on users on the system. It then loops through to access each individual user, and then prints the name (XML tag) and value of each field.

```
var usersXML = system.users;
```

If you uncomment the line of code, recompile, and then run the script again, you can view the suppressed output. It prints all logged-on users as a large XML document.

```
//print( usersXML );
```

This part of the script loops through to process each <user> element within <users>.

```
for ( userIndex in usersXML )
{
  var userXML = usersXML[userIndex];
```

If you uncomment the line of code, recompile, and then run the script again, you can view the suppressed output. It prints an individual user as an XML document.

```
//print( userXML );
```

This part of the script loops through to process each XML element node within the <user> element.

```
for ( fldIndex in userXML )
  {
    var node = userXML[fldIndex];

    print( node.getName() + ": " + node.getValue() );
  }
}
```

Example: Iterating through a complex XML document

This JavaScript example demonstrates how to iterate through a complex XML document that includes any number of levels of nesting and contains attributes. The script visits each node, be it an attribute node or an element node, and prints the name of the node and its value.

```
var xmlObj =  new XML();
xmlObj.setContent( "c:\\test.xml", true );
visitElement( xmlObj.getDocumentElement() );  // getDocumentElement
returns the root node
```

```
function visitElement( elem )
{
 print( "Element " + elem.getQualifiedName() );
 // Print attributes for this element
 var node = elem.getFirstAttribute();
 while( node != null )
 {
   var attrName  = node.getQualifiedName();
   var attrValue = node.getNodeValue();
   print( "Attribute " + attrName + "=" + attrValue );
   node = elem.getNextAttribute();
 }
 // Print possible element value
 var nodeValue = elem.getNodeValue();
 if ( typeof nodeValue == "string" && nodeValue.length > 0 )
 {
    print( "Element value is " + nodeValue );
 }
 // Now print child elements
 var child = elem.getFirstChildElement();
 while( child != null )
 {
   var nodeName  = child.getNodeName();
   var nodeValue = child.getNodeValue();
   visitElement( child );
   child = child.getNextSiblingElement();
 }
}
```

Example: Query a record using the SCFile object

This JavaScript example demonstrates how to perform the following objectives.

1. Query a record from a table using the SCFile object.
2. Obtain the resulting record in XML format by using the getXML method that gives you field names in addition to values.
3. Iterate through the resulting XML and print the field names and values. For a more robust approach, see "Example: Iterating through a complex XML document" on previous page.

```
var fContacts = new SCFile( "contacts" );

var rc = fContacts.doSelect( "contact.name = \"BROWN, NICHOLAS\" " );

if ( rc != RC_SUCCESS )  // doSelect can also return RC_BAD_QUERY or
RC_NO_MORE
{
  print( "Error - doSelect returned " + RCtoString(rc) );
```

```
}
else
{
  var xmlContactsRecord = fContacts.getXML().instance;

  var fldIndex;

  for ( fldIndex in xmlContactsRecord )
  {
    var node  = xmlContactsRecord[fldIndex];

    if ( node.getFirstChildElement() != null ) // this node has child
elements
    {
      var i;

      for ( i in node ) // print the child elements
      {
        var childNode = node[i];

        print( childNode.getName(), "=", childNode.getValue() );
      }
    }
    else
      print( node.getName(), "=", node.getValue() );

  }
}
```

Example: Testing a file name

This JavaScript example demonstrates how to test whether or not a HP Service Manager file name in a string is valid. The script uses a string containing the file name to create a new SCFile object, and then tests the resulting object to see if it is valid. If the file name is valid, it contains the expected properties and methods (such as "doSelect").

```
function testFile( fname )
{
  var undefined;

  var f = new SCFile( fname );

  if ( typeof f == "object" && f.doSelect != undefined )
  {
    print( fname + " is a real file!" );
  }
  else
    print( fname + " is NOT a real file!" );
```

```
}

testFile( "nonexistentfile" );

testFile( "contacts" );
```

Example: Updating a field with the current date and time

This JavaScript example demonstrates how to update a field with the current date and time. The script takes a file object (such as system.vars.$L_file or an SCFile object) and a string containing the name of a datetime field in that file, and updates the indicated field with the current date and time.

This script illustrates several key concepts of using JavaScript in HP Service Manager.

- How to call the RAD filename function from JavaScript.
- How to access a field in a record using an expression of the form file[*fieldName*] where *fieldName* is a string.
- How to test the field's data type.
- How to return JavaScript date objects when you reference Service Manager datetime fields.
- How to assign JavaScript date objects to Service Manager datetime fields.
- How to use SCFile method return codes to test against global properties like RC_SUCCESS.
- How to obtain a localized error message from a return code value using the RCtoString method.
- How to pass traditional query strings that are in Service Manager syntax directly to the doSelect method.
- How to use underscores in JavaScript to reference Service Manager field names containing dots (such as contact.name).

This script demonstrates calling the datetimestamp function. It selects a record from the contacts table and updates the **sysmodtime** field.

```
var f = new SCFile( "contacts" );

f.doSelect( "contact.name = \"BROWN, NICHOLAS\"" );

print( "sysmodtime was: " + f.sysmodtime );

datetimestamp( f, "sysmodtime" );

print( "sysmodtime is now: " + f.sysmodtime );
```

This script demonstrates how passing the wrong kind of field to the datetimestamp function generates an error message.

```
datetimestamp( f, "contact_name" );

function datetimestamp( file, fieldName )
{
  var fileName;
  var today;
  var fieldType;
  var rc;

  fileName = system.functions.filename( file );
  today    = new Date();

  print( "datetimestamp called for file " + fileName + " and field " +
fieldName );

  var d = new SCDatum( file[fieldName] );

  fieldType = d.getType();

  if ( fieldType != "TIME" )
  {
    print( "Error! Field " + fieldName + " is a " + fieldType + "
field!
               This function can only be called with date/time fields!" );
    return false;
  }

  file[fieldName] = today;  // Timestamp the field

  rc = file.doUpdate();

  if ( rc != RC_SUCCESS )
  {
    print( "Error " + RCtoString(rc) + " trying to update " + fileName
+ " record" );
    return false;
  }

  return true;
}
```

Example: Working with structured arrays

This JavaScript example shows you how to work with structured arrays.

Note: The sample "teststructarray" has the following Database Dictionary format fields:

- id character
- arr array
- arr structure
- a number
- b character

This table has an array of type structure called "arr."

Note: To populate the elements of the field, the elements have to be populated in order.

```
var fvar = SCFile("teststructarray");

fvar.id="abc";

fvar.arr[2].a=10;      // Add field 'a' first
fvar.arr[2].b="Test";   // Then add field 'b'

print(fvar);

[C++ object SCFile] - teststructarray={[["abc",{{[, ]},, {[10,
"Test"]}}]}
```

Caution: Populating the fields out of order will cause an error.

For example, this will break the structure:

```
fvar.arr[2].b="Test";   // This will break the structure!
fvar.arr[2].a=10;
```