IBM Rational DOORS
DXL Reference Manual
Release 9.6.1
# Table of Contents

About this manual ..................................................1  
Typographical conventions ........................................1  
Related documentation ............................................2  

Chapter 1  
Introduction ...........................................................3  
Developing DXL programs .........................................3  
Browsing the DXL library .........................................5  
Localizing DXL .......................................................6  
Language fundamentals .........................................7  
Lexical conventions ................................................10  
Constants ..............................................................12  
Identifiers ..............................................................14  
Types .................................................................15  
Declarations ...........................................................15  
Expressions ...........................................................18  
Statements .............................................................20  
Basic functions ......................................................24  

Chapter 2  
New in DXL for Rational DOORS 9.6.1 ..........................29  
Object Management functions ....................................29  
Module Properties ................................................30  
Dialog box functions ..............................................30  
Display control functions .......................................30  
OLE objects ..........................................................31  
General functions ................................................31  
OSLC DXL Services ................................................33  
Timer .................................................................34  
Triggers ..............................................................35  

Chapter 3  
New in DXL for Rational DOORS 9.6 ............................37  
Operating system interface .......................................37  
Mini database explorer ..........................................37  
Modules ..............................................................38  
History ...............................................................38  
Dialog box functions: common element operations ...........39  
Display control: columns .........................................39  
Display control: Layout DXL ....................................40  
HTTP Server ........................................................40  

Chapter 4  
New in DXL for Rational DOORS 9.5 ............................41  
Embedded OLE objects and the OLE clipboard ...............41  
OSLC Link Discovery ..............................................42  
Database properties ...............................................43
Chapter 5  **New in DXL for Rational DOORS 9.4**  49
- Attribute definitions ........................................ 49
- Attribute types ................................................. 50
- Rich text strings ................................................. 51

Chapter 6  **New in DXL for Rational DOORS 9.3**  53
- Converting a symbol character to Unicode .................. 53
- Dialog box functions ........................................... 54
- Operations on type string ..................................... 54
- Embedded OLE objects and the OLE clipboard ................ 56
- OLE information functions .................................... 56
- Discussions ....................................................... 57
- RIF ID ............................................................. 60
- Rational DOORS URLs .......................................... 60
- Filters ............................................................. 61
- Compound Filters ............................................... 63
- Localizing DXL ................................................ 64
- Finding links ................................................... 65
- Links ............................................................. 67

Chapter 7  **New in DXL for Rational DOORS 9.2**  69
- Additional authentication ..................................... 69
- Dialog box updates ............................................. 70
- New constants .................................................. 71
- Partitions updates ............................................... 72
- Requirements Interchange Format (RIF) ....................... 73

Chapter 8  **New in DXL for Rational DOORS 9.1**  83
- Regular Expressions ........................................... 83

Chapter 9  **New in DXL for Rational DOORS 9.0**  85
- Discussions ..................................................... 85
- Discussion Types ............................................... 85
- Properties ....................................................... 86
- Iterators ......................................................... 88
- Operations ...................................................... 90
- Triggers .......................................................... 92
- Example ........................................................ 93
- Descriptions .................................................... 95
- View Descriptions ............................................. 95
- Attribute Type Descriptions .................................. 95
- Attribute Definition Descriptions .............................. 97
- Filtering ........................................................ 98
- HTML ........................................................... 98
Chapter 10  Fundamental types and functions  ........................................ 113
  Operations on all types ................................................................. 113
  Operations on type bool .............................................................. 115
  Operations on type char ............................................................... 116
  Operations on type int ............................................................... 119
  Operations on type real .............................................................. 122
  Operations on type string ........................................................... 126

Chapter 11  General language facilities ............................................. 133
  Files and streams ................................................................. 133
  Configuration file access .......................................................... 141
  Dates ................................................................................. 147
  Skip lists .............................................................................. 154
  Regular expressions ................................................................. 158
  Text buffers ........................................................................... 163
  Arrays .................................................................................. 172

Chapter 12  Operating system interface .............................................. 177
  Operating system commands ...................................................... 177
  Windows registry ................................................................. 184
  Interprocess communications .................................................... 187
  System clipboard functions ....................................................... 190

Chapter 13  Customizing Rational DOORS ....................................... 193
  Color schemes ................................................................. 193
  Database Explorer options ......................................................... 195
  Mini database explorer ............................................................. 198
  Locales .............................................................................. 199
  Codepages ........................................................................ 205
  Message of the day ................................................................. 209
  Database Properties ............................................................... 210

Chapter 14  Rational DOORS database access ..................................... 213
  Database properties ................................................................. 213
  Group and user manipulation .................................................... 230
  Group and user management ................................................... 238
  LDAP ................................................................................ 248
  LDAP Configuration .............................................................. 250
  LDAP server information ........................................................ 252
  LDAP data configuration ........................................................ 256
  Rational Directory Server ......................................................... 260
# Rational DOORS hierarchy

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About the Rational DOORS hierarchy</td>
<td>265</td>
</tr>
<tr>
<td>Item access controls</td>
<td>266</td>
</tr>
<tr>
<td>Hierarchy clipboard</td>
<td>267</td>
</tr>
<tr>
<td>Hierarchy information</td>
<td>270</td>
</tr>
<tr>
<td>Hierarchy manipulation</td>
<td>274</td>
</tr>
<tr>
<td>Items</td>
<td>276</td>
</tr>
<tr>
<td>Folders</td>
<td>279</td>
</tr>
<tr>
<td>Projects</td>
<td>282</td>
</tr>
<tr>
<td>Looping within projects</td>
<td>287</td>
</tr>
</tbody>
</table>

# Modules

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module access controls</td>
<td>289</td>
</tr>
<tr>
<td>Module references</td>
<td>290</td>
</tr>
<tr>
<td>Module information</td>
<td>293</td>
</tr>
<tr>
<td>Module manipulation</td>
<td>297</td>
</tr>
<tr>
<td>Module display state</td>
<td>302</td>
</tr>
<tr>
<td>Baselines</td>
<td>305</td>
</tr>
<tr>
<td>Baseline Set Definition</td>
<td>313</td>
</tr>
<tr>
<td>Baseline Sets</td>
<td>322</td>
</tr>
<tr>
<td>History</td>
<td>331</td>
</tr>
<tr>
<td>Descriptive modules</td>
<td>341</td>
</tr>
<tr>
<td>Recently opened modules</td>
<td>344</td>
</tr>
<tr>
<td>Module Properties</td>
<td>346</td>
</tr>
</tbody>
</table>

# Electronic Signatures

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature types</td>
<td>351</td>
</tr>
<tr>
<td>Controlling Electronic Signature ACL</td>
<td>351</td>
</tr>
<tr>
<td>Electronic Signature Data Manipulation</td>
<td>355</td>
</tr>
<tr>
<td>Examples</td>
<td>360</td>
</tr>
</tbody>
</table>

# Objects

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About objects</td>
<td>369</td>
</tr>
<tr>
<td>Object access controls</td>
<td>369</td>
</tr>
<tr>
<td>Finding objects</td>
<td>371</td>
</tr>
<tr>
<td>Current object</td>
<td>376</td>
</tr>
<tr>
<td>Navigation from an object</td>
<td>377</td>
</tr>
<tr>
<td>Object management</td>
<td>380</td>
</tr>
<tr>
<td>Information about objects</td>
<td>384</td>
</tr>
<tr>
<td>Selecting objects</td>
<td>386</td>
</tr>
<tr>
<td>Object searching</td>
<td>388</td>
</tr>
<tr>
<td>Miscellaneous object functions</td>
<td>389</td>
</tr>
</tbody>
</table>

# Links

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About links and link module descriptors</td>
<td>393</td>
</tr>
<tr>
<td>Link creation</td>
<td>394</td>
</tr>
<tr>
<td>Link access control</td>
<td>394</td>
</tr>
<tr>
<td>Chapter 23</td>
<td>Templates</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td>Template functions</td>
</tr>
<tr>
<td></td>
<td>Template expressions</td>
</tr>
<tr>
<td>Chapter 24</td>
<td>Rational DOORS window control</td>
</tr>
<tr>
<td></td>
<td>The DXL Library and Addins menus</td>
</tr>
<tr>
<td></td>
<td>Module status bars</td>
</tr>
<tr>
<td></td>
<td>Rational DOORS built-in windows</td>
</tr>
<tr>
<td></td>
<td>Module menus</td>
</tr>
<tr>
<td>Chapter 25</td>
<td>Display control</td>
</tr>
<tr>
<td></td>
<td>Filters</td>
</tr>
<tr>
<td></td>
<td>Compound filters</td>
</tr>
<tr>
<td></td>
<td>Filtering on multi-valued attributes</td>
</tr>
<tr>
<td></td>
<td>Sorting modules</td>
</tr>
<tr>
<td></td>
<td>Views</td>
</tr>
<tr>
<td></td>
<td>View access controls</td>
</tr>
<tr>
<td></td>
<td>View definitions</td>
</tr>
<tr>
<td></td>
<td>Columns</td>
</tr>
<tr>
<td></td>
<td>Scrolling functions</td>
</tr>
<tr>
<td></td>
<td>Layout DXL</td>
</tr>
<tr>
<td>Chapter 26</td>
<td>Partitions</td>
</tr>
<tr>
<td></td>
<td>Partition concepts</td>
</tr>
<tr>
<td></td>
<td>Partition definition management</td>
</tr>
<tr>
<td></td>
<td>Partition definition contents</td>
</tr>
<tr>
<td></td>
<td>Partition management</td>
</tr>
<tr>
<td></td>
<td>Partition information</td>
</tr>
<tr>
<td></td>
<td>Partition access</td>
</tr>
<tr>
<td>Chapter 27</td>
<td>Requirements Interchange Format (RIF)</td>
</tr>
<tr>
<td></td>
<td>RIF export</td>
</tr>
<tr>
<td></td>
<td>RIF import</td>
</tr>
<tr>
<td></td>
<td>RIF ID</td>
</tr>
<tr>
<td></td>
<td>Merge</td>
</tr>
<tr>
<td></td>
<td>RIF definition</td>
</tr>
<tr>
<td></td>
<td>Examples</td>
</tr>
<tr>
<td>Chapter 28</td>
<td>OLE objects</td>
</tr>
<tr>
<td></td>
<td>Embedded OLE objects and the OLE clipboard</td>
</tr>
<tr>
<td></td>
<td>OLE information functions</td>
</tr>
<tr>
<td></td>
<td>Picture object support</td>
</tr>
<tr>
<td></td>
<td>Automation client support</td>
</tr>
<tr>
<td></td>
<td>Controlling Rational DOORS from applications that support automation</td>
</tr>
<tr>
<td>Chapter 29</td>
<td>Triggers</td>
</tr>
</tbody>
</table>
### Chapter 30: Page setup functions
- Page attributes status ........................................ 805
- Page dimensions .............................................. 806
- Document attributes ......................................... 809
- Page setup information ....................................... 812
- Page setup management ..................................... 815

### Chapter 31: Tables
- Table concept .................................................. 817
- Table constants ............................................... 817
- Table management ........................................... 818
- Table manipulation ........................................... 822
- Table attributes .............................................. 830

### Chapter 32: Rich text
- Rich text processing .......................................... 833
- Rich text strings ............................................... 840
- Enhanced character support ................................. 852
- Importing rich text .......................................... 855
- Diagnostic perms ............................................. 856

### Chapter 33: Spelling Checker
- Constants and general functions ......................... 861
- Language and Grammar .................................... 870
- Spelling Dictionary .......................................... 879
- Miscellaneous Spelling .................................... 882
- Spelling\Dictionary Examples .............................. 884

### Chapter 34: Database Integrity Checker
- Database Integrity Types .................................. 887
- Database Integrity Perms .................................. 888

### Chapter 35: Discussions
- Discussion Types ............................................. 897
- Properties ..................................................... 897
- Iterators ....................................................... 900
- Operations .................................................... 901
- Triggers ....................................................... 905
- Discussions access controls ............................... 906
- Example ....................................................... 908
# Chapter 36  General functions ................................................................. 911
  Error handling .................................................................................. 911
  Archive and restore ................................................................. 914
  Checksum validation ........................................................................ 927
  Locking ............................................................................................... 931
  HTML functions ................................................................................ 938
  HTTP Server .................................................................................... 940
  Asynchronous HTTP requests ....................................................... 945
  OSLC DXL Services .......................................................................... 949
  Broadcast Messaging ....................................................................... 951
  Converting a symbol character to Unicode .................................... 952
  Timer ................................................................................................. 953

# Chapter 37  Character codes and their meanings ................................. 957

# Chapter 38  Notices ................................................................. 963

# Index ......................................................................................... 965
About this manual

Welcome to version 9.6.1 of IBM® Rational® DOORS®, a powerful tool that helps you to capture, track and manage your user requirements.

DXL (DOORS eXtension Language) is a scripting language specially developed for Rational DOORS. DXL is used in many parts of Rational DOORS to provide key features, such as file format importers and exporters, impact and traceability analysis and inter-module linking tools. DXL can also be used to develop larger add-on packages such as CASE tool interfaces and project management tools. To the end user, DXL developed applications appear as seamless extensions to the graphical user interface. This capability to extend or customize Rational DOORS is available to users who choose to develop their own DXL scripts.

The DXL language is for the more technical user, who sets up programs for the end-user to apply. DXL takes many of its fundamental features from C and C++. Anyone who has written programs in these or similar programming languages should be able to use DXL.

This book is a reference manual for DXL for version 9.6.1 of Rational DOORS. Refer to it if you wish to automate simple or complex repetitive tasks, or customize your users’ Rational DOORS environment. It assumes that you know how to write C or C++ programs.

Typographical conventions

The following typographical conventions are used in this manual:

<table>
<thead>
<tr>
<th>Typeface or Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Important items, and items that you can select, including buttons and menus: “Click <strong>Yes</strong> to continue”.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Book titles.</td>
</tr>
<tr>
<td><strong>Courier</strong></td>
<td>Commands, files, and directories; computer output: “Edit your .properties file”.</td>
</tr>
<tr>
<td>&gt;</td>
<td>A menu choice: “Select <strong>File</strong> &gt; <strong>Open</strong>”. This means select the <strong>File</strong> menu, and then select the <strong>Open</strong> option.</td>
</tr>
</tbody>
</table>

Each function or macro is first introduced by name, followed by a declaration or the syntax, and a short description of the operation it performs. These are supplemented by examples where appropriate.
Related documentation

The following table describes where to find information in the Rational DOORS documentation set:

<table>
<thead>
<tr>
<th>For information on</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rational DOORS</td>
<td>The Rational DOORS Information Center</td>
</tr>
<tr>
<td>How to set up licenses to use Rational DOORS</td>
<td>Rational Lifecycle Solutions Licensing Guide</td>
</tr>
<tr>
<td>How to write requirements</td>
<td>Get It Right the First Time</td>
</tr>
<tr>
<td>How to integrate Rational DOORS with other applications</td>
<td>Rational DOORS API manual</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter describes the DXL Interaction window, DXL library, and the basic features of DXL. It covers the following topics:

- Developing DXL programs
- Browsing the DXL library
- Localizing DXL
- Language fundamentals
- Lexical conventions
- Constants
- Identifiers
- Types
- Declarations
- Expressions
- Statements
- Basic functions

Developing DXL programs

You can use the DXL Interaction window to develop small DXL programs.

For large-scale program development, you should use a third party editing tool when coding, and then load your code into the DXL Interaction window to execute and debug it. You can set up a menu option in Rational DOORS to run your third party editing tool.
To use the DXL Interaction window:

1. In either the Database Explorer or a module window, click **Tools > Edit DXL**.

2. Either type or load your program into the DXL input pane.

   To load the contents of a file, click **Load**. To load a program from the DXL library, click **Browse**.

3. To run the program in the DXL input pane, click **Run**.

   Any error messages that are generated are displayed in the DXL output pane.

   To see the next error message, click **Next error**. The contents of the DXL input pane scroll to the line of source code that caused the error displayed in the DXL output pane.

4. To print the contents of the DXL input pane with line numbers, click **Print**.

5. To save the contents of the DXL input pane to file, click **Save As**.
Right-click anywhere in the DXL input pane to display a pop-up menu with the sub-menus **File**, **Edit**, and **Search**. The **Edit** sub-menu options have standard Windows functions. The **File** sub-menu options are described in the following table:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>Loads the contents of a text file into the DXL input pane. You can also use drag-and-drop to load a file directly from Windows Explorer.</td>
</tr>
<tr>
<td>Save</td>
<td>Saves changes you made to the text in the DXL input pane.</td>
</tr>
<tr>
<td>Save as</td>
<td>Saves the contents of the DXL input pane to another file.</td>
</tr>
<tr>
<td>New</td>
<td>Clears the DXL input pane. If you have made changes to the text that have not yet been saved, you are asked if you want to save them.</td>
</tr>
</tbody>
</table>

The **Search** sub-menu options are described in the following table:

<table>
<thead>
<tr>
<th>Search</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>Finds a string of text in the DXL input pane. The search is case-sensitive.</td>
</tr>
<tr>
<td>Again</td>
<td>Repeats the search.</td>
</tr>
<tr>
<td>Replace</td>
<td>Replaces one string of text with another. You can replace text strings one at a time or all at once.</td>
</tr>
<tr>
<td>Goto line</td>
<td>Moves the cursor to the start of a specified line. (This is useful when debugging DXL programs because errors are indicated against line numbers.)</td>
</tr>
</tbody>
</table>

## Browsing the DXL library

The DXL library is in the `/lib/dxl` folder in the Rational DOORS home directory.

You can browse the DXL library when you are:

- Using the DXL Interaction window, by clicking the **Browse** button to find a program to run.
- Creating a DXL attribute, by clicking the **Browse** button to find a program to use for the attribute (see “DXL attribute,” on page 465).
- Creating a layout DXL column, by clicking the **Browse** button to find a program to use for the layout DXL column (see “Layout DXL,” on page 698).
You see the DXL Library window. The DXL programs and the buttons you see depend on where you were when you clicked the Browse button.

<table>
<thead>
<tr>
<th>Button</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Runs the selected program in your DXL Interaction window.</td>
</tr>
<tr>
<td>Edit</td>
<td>Edits the selected program.</td>
</tr>
<tr>
<td>Print</td>
<td>Prints the selected program.</td>
</tr>
</tbody>
</table>

Localizing DXL

Rational DOORS uses ICU resource bundles for accessing translated strings. DXL perms are available to access ICU resource bundles containing translated strings for customized DXL. For information about creating ICU resource bundles, see http://userguide.icu-project.org/locale/localizing.

Put the language resource files in a directory whose name is taken as the bundle name, under $DOORSHOME/language, for example $DOORSHOME/language/myResource/de_DE.res. There are two bundles already shipped with Rational DOORS, core and DXL.)
Declaration

string LS_(string key, string fallback, string bundle)

Operation

Returns the string from resource bundle that is identified by key. If the string identified by key is not found in the resource bundle, the fallback string is returned.

Example

dev.txt file contains;
de {
    Key1{"Ausgehend"}
    Key2{"Ausgehende Links"}
    Key3{"Normalansicht"}
    Key4{"Klartext"}
}

From the command line, generate a resource bundle, for example genrb dev.txt, and copy the resource bundle to $DOORHOME/language/myResource/, where myResource is the name of your resource bundle. The localized strings can then be accessed using the LS_perm, for example in the DXL editor, type:

print LS_("Key1", "Ausgehend not found", "myResource") "\n"
print LS_("Key2", "Ausgehende Links not found", "myResource") "\n"
print LS_("Key3", "Normalansicht not found", "myResource") "\n"
print LS_("Key4", "Klartext not found", "myResource") "\n"

The output is:
Ausgehend
Ausgehende Links
Normalansicht
Klartext

Language fundamentals

DXL is layered on an underlying programming language whose fundamental data types, functions and syntax are largely based on C and C++. To support the needs of script writing, there are some differences. In particular, concepts like main program are avoided, and mandatory semicolons and parentheses have been discarded.
Auto-declare

In DXL there is a mechanism called auto-declare, which means that a user need not specify a type for a variable. For example, in the script:

```dxl
i=5
print i
```

the interpreter declares a new variable and deduces from the assignment that its type is `int`.

Because DXL is case-sensitive, there is a potential hazard when relying on this mechanism to type variables. If you make a mistake when typing a variable name, the interpreter assumes that a new variable is being used, which creates errors that are hard to find.

This feature can be disabled by adding the line:

```dxl
XFLAGS_ &=~AutoDeclare_
```

to the bottom of the file `$DOORSHOME/lib/dxl/startup.dxl`.

Syntax

The syntactic style is more like natural language or standard mathematical notation. Consider the function:

```dxl
string deleteUser(string name)
```

This can be called as follows:

```dxl
deleteUser "Susan Brown"
```

The lack of semicolons is possible through DXL’s recognition of the end of a line as a statement terminator, except when it follows a binary operator. This means you can break an expression like `2+3` over a line by making the break after the `+` sign.

A comment ending in a dash (`//`) also enables line continuation.

As in C, `==` is used for equality, while `=` is used for assignment. Unlike C or Pascal, concatenation of symbols is a valid operation.

Parsing

Statement or expression parsing is right associative and has a relatively high precedence. Parenthesis has the highest precedence.

Because `sqrt` is defined as a function call that takes a single type real argument:

```dxl
sqrt 6.0
```

is recognized as a valid function call, whereas in C it is:

```dxl
sqrt(6.0)
```

So, the C statement:

```dxl
print(sqrt(6.0))
```

can be:
print sqrt 6.0

in DXL.

The following script declares a function max, which takes two type int arguments:

```dxl
int max(int a, b) {
    if a < b then return b else return a
}
```

print max(2, 3)

The call of max is parsed as `print(max(2,3))`, which is valid. The statement:

```dxl
print max 2,3
```

would generate errors. Because the comma has a lower precedence than concatenation, it is parsed as:

```dxl
((print max(2)),3)
```

If in doubt, use the parentheses, and separate statements for concatenation operations.

---

**Naming conventions**

As a general rule, DXL reserves identifiers ending in one or more underscores (\_, \__) for its own use. You should not use functions, data types or variables with trailing underscores, with the exception of those documented in this manual.

Names introduced as data types in DXL, such as `int`, `string`, `Module` and `Object`, must not be used as identifiers. The fundamental types such as `int` and `string` are in lower case. Rational DOORS specific types all start with an upper case letter to distinguish them from these, and to enable their lower case versions to be used as identifiers.

---

**Loops**

In DXL, loops are treated just like any other operator, and are **overloaded**, that is, declared to take arguments and return values of more than one type. The loop notation used is as follows:

```dxl
for variable in something do {
    ...
}
```

The for loops all iterate through all values of an item, setting variable to each value in turn.

**Note:** When using for loops, care must be taken when deleting items within the loop and also opening and closing items within a for loop. For example, if `variable` is of type `Module` and `something` is of type `Project`, and within the for loop a condition is met that means one of the modules will be deleted, this should not be done within the for loop as it can lead to unexpected results. A recommended method is to use a skip list to store the modules and to do any manipulation required using the contents of the skip list.
Lexical conventions

Semicolon and end-of-line

DXL diverges from C in that semicolons can be omitted in some contexts, with end-of-line (newline) causing statement termination. Conversely, newline does not cause statement termination in other contexts. This is a useful property; programs look much better, and in practice the rules are intuitive. The rules are:

• Any newlines or spaces occurring immediately after the following tokens are ignored:

\[
; , ? : = ( + * [ \& \- ! ~ / % << >> <*> <>
< > < = > = == != ^ | &&
and || or ^^ += -= *= /= %=
<== >>= &= |= ^= <- := => . .
. -> :: \.
\]

• Any newlines before an else or a ) are ignored. All other newlines delimit a possibly empty statement.

• Multiple consecutive areas of white space containing newlines are treated as single newlines.

• The recognition of a newline can be avoided by prefixing it with an empty // comment or a comment ending in -.

Comments

The characters /* start a comment that terminates with the characters */. This style of comment does not nest.

The characters // start a comment that terminates at the end of the line on which it occurs. The end-of-line is not considered part of the comment unless the comment is empty or the final character is -. This latter feature is useful for adding comments to a multi-line expression, or for continuing a concatenation expression over two lines.

Notably, comments that immediately follow conditional statements can cause code to behave unexpectedly.

The following program demonstrates some comment forms:

/* Some comment examples (regular C comment) */
int a   // a C++ style comment
int b = 1 +   // We need a trailing - at the end       -
   2   // to prevent a syntax error between "+" and the newline
print //
   "hello" // the // after print causes the following newline to be
   // ignored
/*
Identifiers

An identifier is an arbitrarily long sequence of characters. The first character must be a letter; the rest of the identifier may contain letters, numerals or either of the following two symbols: 

_,

DXL is case sensitive (upper- and lower-case letters are considered different).

The following words are reserved for use as keywords, and must not be used otherwise:

and  bool  break  by  case  char
const continue default do else enum
for if in int module object
or pragma real return sizeof static
struct string switch then union void
while

The following keywords are not currently supported in user programs, but are reserved for future use:

case  const  default  enum
struct  switch  union

A keyword is a sequence of letters with a fixed syntactic purpose within the language, and is not available for use as an identifier.

File inclusion

To include files into DXL scripts, you can use either of the following:

#include "file"
#include <file>

Absolute or relative path names can be used. Relative paths must be based on one of the following forms depending on the platform:

$DOORSHOME/lib/dxl (UNIX)
$DOORSHOME\lib\dxl (Windows)
where DOORSHOME is defined in a UNIX® environment variable, or on Windows platforms in the registry. The Windows-style file separator (\) must be duplicated so that DXL does not interpret it as a meta-character in the string.

If the addins directory is defined in a UNIX environment variable or the Windows registry, this directory is also searched, so relative path names can be with respect to the addins directory.

**Note:** The UNIX shell file name specification form ~user/ is not supported.

### Pragmas

Pragmas modify the background behavior of the DXL interpreter, for example:

```dxl
pragma runLim, int cyc
```

sets the timeout interval `cyc` as a number of DXL execution cycles. The timeout is suppressed if `cyc` is set to zero, as shown in the following example:

```dxl
pragma runLim, 0       // no limit
pragma runLim, 1000000 // explicit limit
```

There is also a pragma for setting the size of the DXL runtime stack, which is used as follows:

```dxl
pragma stack, 10000
```

The default value is set to 1,000,000.

If running the DXL from the DXL editor, when the timeout limit is reached a message is displayed asking if you want to:

- Continue - script execution continues with the same timeout limit.
- Continue doubling the timeout - script execution continues with double the current timeout limit.
- Halt execution - DXL is halted with a run-time error.

If running in batch mode, it is good practise to execute scripts in the DXL editor initially to detect any errors or timeouts. Pragma runLim, 0 should be used in instances of timeouts.

### Constants

#### Integer constants

An integer constant consisting of a sequence of digits is interpreted as octal if it begins with a 0 (digit zero); otherwise it is interpreted as decimal.

A sequence of digits preceded by 0x or 0X is interpreted as a hexadecimal integer.

A sequence of 0s or 1s preceded by 0b is interpreted as a binary number, and converted to an integer value.
Character constants

A character constant is a character enclosed in single quotes, as in 'x'. The value of a character constant is defined to be of type char.

Certain non-graphic characters, the single quote and the backslash, can be represented according to the following escape sequences:

<table>
<thead>
<tr>
<th>Character</th>
<th>Escape sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>newline</td>
<td>\n</td>
</tr>
<tr>
<td>horizontal tab</td>
<td>\t</td>
</tr>
<tr>
<td>backspace</td>
<td>\b</td>
</tr>
<tr>
<td>carriage return</td>
<td>\r</td>
</tr>
<tr>
<td>form-feed</td>
<td>\f</td>
</tr>
<tr>
<td>backslash</td>
<td>\</td>
</tr>
<tr>
<td>single quote</td>
<td>'</td>
</tr>
<tr>
<td>bit pattern</td>
<td>\ddd</td>
</tr>
<tr>
<td>any other character</td>
<td>\c</td>
</tr>
</tbody>
</table>

The escape \ddd consists of the backslash followed by 1, 2, or 3 octal digits.

Any other character that is escaped is passed straight through.

Type real constants

A type real consists of an integer part, a decimal point, a fraction part, an e or E, and an integer exponent. The integer and fraction part both consist of a sequence of digits.

You can omit either the integer part or the fraction part, but not both. You can omit either the decimal point or the exponent with its e or E. You can add a sign to the exponent.

Example

1.0
0.1
1e10
1.2E30
The null constant

The constant null is used as a polymorphic value to indicate a null value. You can use it for any derived type (see “Derived types,” on page 15). You can use it for both assignment to variables and conditional tests on variables.

Example

Object obj = null
if (null obj) {
    ack "This object is empty"
}

Strings

A string literal, of type string and storage class static, is a sequence of characters surrounded by double quotes, as in "apple".

Within a string the double quote (") must be preceded by a backslash (\). For example “Pear\’” is the string Pear’ in quotes. In addition, you can use the same escape sequences as described in “Character constants,” on page 13, including the newline character.

Identifiers

Identifiers denote variables, functions, types and values. You can introduce an identifier into a program by declaration or by immediate declaration. Immediate declaration is when an undeclared identifier is used as the left hand side of an assignment statement.

Variables

Variables represent regions of computer memory. The meaning of the value stored in a variable is determined by the type of the identifier used to access the variable.

Unassigned variables contain the unassigned pattern, which is checked on all references. In this way, errors with unassigned variables are avoided, and an accurate error message is reported.

Scope

Once declared, an identifier has a region of validity within the program known as its scope.

In general, identifiers are in scope following their declaration within the current block, and are available within nested blocks. Identifiers can be hidden by re-declaration in nested blocks. For example, the following code prints a 4 and then a 3 in the output pane of the DXL Interaction window.

```
int i = 3
```
if (true){
    int i = 4
    print i "\n"
}
print i "\n"

Types

Fundamental types

DXL has the following base types:

<table>
<thead>
<tr>
<th>Base type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>Denotes the domain of values true and false, which are provided as predefined constants.</td>
</tr>
<tr>
<td>char</td>
<td>Is similar to the C character type.</td>
</tr>
<tr>
<td>int</td>
<td>Is the only integer type provided in DXL. On all platforms, integers are signed, and have a precision of 32 bits.</td>
</tr>
<tr>
<td>real</td>
<td>Is like the double type in C, with a precision of 64 bits.</td>
</tr>
<tr>
<td>void</td>
<td>Is the type with no values; its main use is in declaring functions that do not return a result.</td>
</tr>
<tr>
<td>string</td>
<td>Is similar to the derived C type char*.</td>
</tr>
</tbody>
</table>

Derived types

DXL supports arrays, functions and references. An internal class facility provides new non-fundamental types, referred to as built-in types, such as Object, Module and Template. DXL does not support class creation by user programs.

Declarations

Declarations are the mechanism used to associate identifiers with variables, functions or values.

Declarators

DXL follows C in its declarator syntax. However, only the simple forms should be necessary in DXL programs.
DXL extends C style arrays by enabling a variable to define the bounds of the array. The number of elements in an array is available by using the `sizeof` function.

Unlike C, DXL arrays can have only one dimension.

In addition to the normal C declarator forms, DXL provides the C++ reference declarator `&`.

DXL uses the ANSI C method of supplying a function’s formal parameters in the declarator itself with each argument given as a fully specified type.

The following script gives some example declarations:

```dxl
int i, j, k     // declare 3 integers
int n = 4       // declare an integer and initialize it
bool a[2]       // declare an array of type bool of size 2
int b[n]        // declare an integer array of size n
print sizeof a  // prints "2"
```

**Note:** A declaration of the form ‘int n = {1,2,3}’ is not supported.

### Immediate declaration

Immediate declaration is a DXL extension from C, which means that the first use of an undeclared variable is also a declaration. It must be used in a context where an unambiguous value is given to it, for example the left hand side of an assignment statement:

```dxl
i = 2
print i
```

Once declared, the identifier must be used consistently.

### Function definitions

DXL functions are very close to the style of ANSI C functions. The following script gives some examples:

```dxl
// define a function to find the maximum of two integers
int n
int max(int a, b) {  
    return a < b ? b : a
}  // max

// This function applies f to every element in a,
// using an accumulation variable r that is initialized to base.
int apply_accumulate(int base, int a[], int f(int, int)) {  
    int r = base
    for (i = 0; i < sizeof a; i++) {  
        r = f(r, a[i])
    }
```
DXL Reference Manual

17

return r
}  // apply_accumulate
int a[5]
print "Filling an array:\n\n"
for (i = 0; i < sizeof a; i++) {
    a[i] = random 1000
    print a[i] "\n"
}  // for
print "largest number was: "
print apply_accumulate(0, a, max)
// print largest element in a

Line 3 defines the function max, which has two parameters of type int and returns a type int. One difference from ANSI C is that the parameter type specifier int need not be repeated before the b parameter.

Line 10 declares a function parameter f. Note that f's parameters do not include redundant identifiers.

Operator functions

You can redefine DXL operators by prefixing the operator with :: to turn it into an identifier.

Example

This example defines a multiplication operator that applies to strings and integers.

```dxl
string ::*(string s, int n) {
    string x = ""
    int i
    for i in 0 : n-1 do {
        x = x s
    }
    return x
}
print ("apple " * 4)
```

This prints out:

apple apple apple apple

If you wish to overload the concatenation operator, which is normally represented by a space, use the symbol ...

```dxl
string ::..(real r, int n) {
    string s = ""
    int i
    // concatenate the string to a space n times
```
for i in 0:n-1 do {
    s=s r " "
}
return s
}
print (2.45 3) "\n" // try it out

The program prints the string:
2.450000 2.450000 2.450000

Expressions

This section outlines the major differences between C and DXL expressions. The operations defined on DXL fundamental types are explained in “Fundamental types and functions,” on page 113.

Reference operations

DXL supports C++ style reference operations. References are like var parameters in Pascal or Ada, which means they provide an alias to a variable, not a copy. To declare a reference variable its name must be preceded by an ampersand (&).

Example

This example is a program to swap two integers. In C you have explicitly to pass the address of the variables to be swapped and then de-reference them within the body of the function. This is not required in DXL.

// swap two integers
void swap (int &a, &b) {
    int temp
    temp = a; a = b; b = temp
}
int x = 2
int& z = x // z is now an alias for x
int y = 3
print " " y "\n"
swap(z, y) // equivalent to swap(x,y)
print " " y "\n"

This program prints the string:
2 3
3 2
Overloaded functions and operators

Most functions and operators can be declared to take arguments and return values of more than one type.

Example
This example overloads a commonly used identifier print to provide an object printer.

```dxl
// Overload print to define an Object printer
void print(Object o) {
    string h = o."Object Heading"
    string t = o."Object Text"
    print h "

    t "
}
print current Object
```

Function calls

DXL enables calls of functions defined without parameters to omit the empty parenthesis, except where the call appears as a function argument or any other context where a function name is valid. Function calls with single arguments can also omit the parenthesis, but beware of concatenation’s high precedence when the argument passed is an expression.

**Note:** When overloading functions, ensure that the first declaration of the function does not have a void parameter, e.g. `void print(void)`. This may lead to unexpected results. Furthermore, function calls of the form `void print(int i=0, int g=0)` should also not be used.

Example

```dxl
void motto() { // parameterless
    print "A stitch in time saves nine."
} // motto
int square(int x) {
    return x*x
} // square
motto // call the function
print square 9 // two function calls
```

Casts

Because of DXL’s overloading facility, it is easy to write expressions that have more than one possible interpretation; that is, they are ambiguous. **Casts** are used to pick which interpretation is required. Casts in DXL come in two forms:

```
expression type
```
(type expression)
In the first form, the type name can appear after the expression, as in:
\[ o = \text{current Object} \]
In the second form, the type may come first, but the whole expression must be within parenthesis:
\[ o = (\text{Object current}) \]

Range

A range expression extracts a substring from a string, or substring from a buffer, and is used in regular expression matching. It has two forms:

- int from : int to
- int from : int to by int by

Examples are given with the functions that use ranges.

Statements

This section describes how to construct statements in DXL.

Compound statements

Compound statements are also referred to as blocks.
Several statements can be grouped into one using braces \{ ... \}.

Conditional statements

The if statement takes an expression of type bool, which must be in parenthesis. If the expression evaluates to true, it executes the following statement, which can be a block. If the expression evaluates to false, an optional else statement is executed.

As an alternative form, the parenthesis around the condition can be dropped, and the keyword then used after the condition.

Example

\[
\begin{align*}
\text{int } i &= 2, \ j = 2 \\
\text{if } (i < 3) \{ \\
\quad i &= +2 \\
\} \text{ else } \{ \\
\quad i &= +3 \\
\} 
\end{align*}
\]
if i == j then j = 22
The then form does not work with a condition that starts with a component in parenthesis, for example:
if (2 + 3) == 4 then print "no"
generates a syntax error.
DXL also supports the C expression form:
2 + 3 == 5 ? print "yes" : print "no"

The if statement also supports multiple else if statements, which can be written as elseif.

Example

int i = 1, j = 2

if (i < j) {
    i += 3
} else if (i == j) {
    i += 2
} else if (i > j) {
    i += 1
}

Do not add a comment in the middle of the else if statement.

Example

int i = 1, j = 2

if (i < j) {
    i += 3
} else
    // Do not add comment here
if (i == j) {
    i += 2
{ else if (i > j) {
    i += 1
}

DXL considers the else if to be one statement and can give incorrect results if a comment line with a preceding space or tab is in-between the else and if.

---

### Loop statements

DXL has three main loop (iteration) statements. It supports the C forms:

- `for (init; cond; increment) statement`
- `while (cond) statement`
- and a new form:

```
for type1 v1 in type2 v2 do
```

where type1 and type2 are two types, possibly the same; v1 is a reference variable and v2 is a variable, which can be a range expression (see “Range,” on page 20). This form is heavily used in DXL for defining type-specific loops.

**Example**

```dxl
int x
int a=2
int b=3
for (x=1; x <= 11; x+=2) {
    print x
}
while (a==2 and b==3) {
    print "hello\n";
    a = 3
}
for x in 1 : 11 by 2 do {
    print x
}
```

In this example, the first loop is a normal C for loop; the second is a normal C while loop. Note that DXL offers the keyword `and` as an alternative to `&&`.

The last form in the example uses a range statement, which has the same semantics as the first C-like loop.

---

### Break statement

The `break` statement causes an immediate exit from a loop. Control passes to the statement following the loop.

**Example**

```dxl
int i = 1
```
while (true) {
    print i++
    if (i==10) {
        break
    } // if (i==10)
} // while (true)

---

**Continue statement**

The `continue` statement effects an immediate jump to the loop’s next test or increment statement.

**Example**

```dxl
int i = 1
while (true) {
    if (i==4) { // don't show 4
        i++
        continue
    } // if (i==4)
    print i++
    if (i==10) {
        break
    } // if (i==10)
} // while (true)
```

---

**Return statement**

The `return` statement either exits a void function, or returns the given value in any other function.

**Note:** Care should be taken when using the return statement. For example, assigning a value to a variable where the assignment is a function, and that function returns no value, can lead to unexpected values being assigned to the variable.

**Example**

```dxl
// exit void function
void print(Object o) {
    if (null o)
        return string h = o."Object Heading"
        print h "\n"
} // print
// return given value
```
int double(int x) {
    return x + x      // return an integer
} // double

print double 111

Null statement

The null (empty) statement has no effect. You can create a null statement by using a semicolon on its own.

Example

int a = 3
if (a < 2) ; else print a

Basic functions

This section defines some basic functions, which can be used throughout DXL.

of

This function is used as shown in the following syntax:

    of(argument)

Returns the passed argument, which can be of any type. It has no other effect. It is used to clarify code.

Example

if end of cin then break

sizeof

This function is used as shown in the following syntax:

    sizeof(array[])

Returns the number of elements in the array, which can be of any type.

Example

string strs[] = {"one", "two", "three"}
int ints[] = {1, 2, 3, 4}
print sizeof strs      // prints 3
print sizeof ints      // prints 4
halt

Declaration

```c
void halt();
```

Operation

Causes the current DXL program to terminate immediately. This is very useful if an error condition is detected in a program.

Example

```c
if (null current Module) {
    ack "program requires a current module"
    halt
}
```

checkDXL

Declaration

```c
string checkDXL[File](string code)
```

Operation

Provides a DXL mechanism for checking DXL code.

The `checkDXL` function analyzes a DXL program and returns the string that would have been produced in the DXL Interaction window had it been run on its own.

The `checkDXLFile` function analyzes a file and returns the error message that would have been produced in the DXL Interaction window had the file been run.

Example

```c
string errors =
    checkDXL("int j = 3 \n  print k + j")
if (!null errors)
    print "Errors found in dxl string:
    " errors
"\n"
```

would produce the following in the DXL Interaction window’s output pane.

Errors found in dxl string:
- E- DXL: <Line:2> incorrect arguments for (+)
- E- DXL: <Line:2> incorrect arguments for function (print)
- E- DXL: <Line:2> undeclared variable (k)
sort

Declaration
void sort(string stringArray[])

Operation
Sorts the string array stringArray. The sort function handles string arrays containing non-ASCII characters, as do the string and Buffer comparison operators.

Example
int noOfHeadings = 0
Object o
for o in current Module do {
    string oh = o."Object Heading"
    if (!null oh) noOfHeadings++
}
string headings[noOfHeadings]
int i = 0
for o in current Module do {
    string oh = o."Object Heading"
    if (!null oh) headings[i++] = oh
}
sort headings
for (i = 0; i < noOfHeadings; i++) print headings[i] "\n"

activateURL

Declaration
void activateURL(string url)

Operation
This is equivalent to clicking on a URL in a formal module.

batchMode, isBatch

Declaration
bool batchMode()
bool isBatch()
Operation

Both functions return `true` if Rational DOORS is running in batch mode, and `false` if Rational DOORS is running in interactive mode.
Chapter 2

New in DXL for Rational DOORS 9.6.1

This chapter describes changes to the DXL Reference Manual in Rational DOORS 9.6.1:

- **Object Management functions:**
  - purgeObject_

- **Dialog box functions:**
  - helpOn
  - minimumSize
  - listView

- **Display control functions:**
  - Compound filters
  - Columns: backgroundColor(get)
  - Columns: backgroundColor(set)
  - Layout DXL: setRefreshDelta

- **OLE objects**
  - oleInsert (insert to buffer)

- **General functions:**
  - Checksum validation
  - HTML help
  - Asynchronous HTTP requests
  - OSLC DXL Services
  - Timer
  - Triggers

---

Object Management functions

---

purgeObject_

The function “purgeObject_” on page 384 removes the specified soft-deleted object. Once executed, this object cannot be recovered. The name ends in "_" to discourage casual use. The documentation for this function is added in version 9.6.1.3 of this manual.
Module Properties

delete(ModuleProperties)

The function “delete(ModuleProperties)” on page 347 deletes the supplied moduleProperties structure. If not called after a call to getProperties, the memory will only be released after the context is released. This function is new in Rational DOORS version 9.6.1.4.

Dialog box functions

helpOn

The documentation for the helpOn function has been removed from this manual because the help is no longer delivered in the HTML Help (chm) format.

minimumSize

The new function “minimumSize” on page 500 sets the minimum size of the dialog box to a specified width and height.

listView

Beginning in Rational DOORS version 9.6.1.7, you can use Ctrl+A keys to select all items in a multiselect listView. See “listView” on page 547.

Display control functions

Compound filters

The documentation for “Compound filters” on page 656 includes new example code in version 9.6.1.3 of this manual.
Columns: backgroundColor(get)

The function “backgroundColor(get)” on page 689 returns the name of the attribute that is used to color the background of a specific column. The function was first included in version 9.4.0. The documentation is added in version 9.6.1.3 of this manual.

Columns: backgroundColor(set)

The function “backgroundColor(set)” on page 690 sets the background color of a specific column. The function was first included in version 9.4.0. The documentation is added in version 9.6.1.3 of this manual.

Layout DXL: setRefreshDelta

The function “setRefreshDelta” on page 702 was a new function in Rational DOORS 9.6.0. The documentation is added in version 9.6.1 of this manual.

OLE objects

oleInsert (insert to buffer)

The function “oleInsert (insert to buffer)” on page 742 inserts OLE into the given buffer at a given character offset. The documentation for this function is added in version 9.6.1.4 of this manual.

General functions

Checksum validation

The checksum validation functions enable you to create a validation record for a module before you export or archive the module. When the module is later imported or restored to the project, you can compare the checksum validation record with the module to identify changes to the text or other attribute values in the selected views.

createChecksumFile

The function “createChecksumFile” on page 927 creates a compressed file with a .zip extension that contains XML files that describe a module, selected views within the module, and object attributes that are associated with objects in those views.
loadChecksumFile

The function “loadChecksumFile” on page 928 loads a checksum package for a specific module and loads the list of views that are available for checksum comparison.

compareChecksumFile

The function “compareChecksumFile” on page 929 enables the user to compare the checksum validation record with the related module to identify changes to the text or other attribute values in the selected views.

HTML help: helpOnEx

The documentation for the helpOnEx function has been removed from this manual because the help is no longer delivered in the HTML Help (chm) format.

Asynchronous HTTP requests

The “Asynchronous HTTP requests” on page 945 are used to make HTTP requests asynchronously so that the main Rational DOORS process is not blocked. With these functions, the DXL user interface is still be responsive while an HTTP request is in progress.

Future HttpRequest

The Future HttpRequest function starts an HTTP request but instead of waiting for a response, it immediately returns a Future object. The Future object contains a value that will be delivered in future. The Future object can be checked for readiness later. The HTTP response can be fetched from the Future object when it is ready.

The documentation for this function is added in version 9.6.1.6 of this manual.
OSLC DXL Services

OSLC DXL Services are DXL scripts that can be run by making an Open Services for Lifecycle Collaboration (OSLC) request to an instance of Rational DOORS Web Access. A service must be added to the DOORS database before it can be run. DXL functions are available to add, remove, and return information about DXL services. For more information and examples, see the help topic: OSLC DXL services for Rational DOORS.

Support for OSLC DXL Services was added to Rational DOORS in a previous release. This documentation is added in version 9.6.1.4.

OSLCDXLService properties

“OSLCDXLService properties” on page 949 are defined for use with the . (dot) operator and a OSCDXLService handle.

setDxlServiceResult

The function “setDxlServiceResult” on page 950 sets the result string that is returned as a result of the service that is being run.

addOrUpdateOSLCDXLService

The function “addOrUpdateOSLCDXLService” on page 950 adds a new service to the list of configured DXL services or updates an existing one.

removeOSLCDXLService(string key)

The function “removeOSLCDXLService(string key)” on page 951 removes a service from the configuration. Finds the service by its key (that is, name).

removeOSLCDXLService(OSLCDXLService service)

The function “removeOSLCDXLService(OSLCDXLService service)” on page 951 removes a service object from the configuration.
Timer

Documentation for the timer functions is added in version 9.6.1.4. The functions are available in earlier versions.

timer

“Timer” on page 953 creates a timer element that executes the callback function every 'n' seconds.

stopTimer

The function “stopTimer” on page 953 stops the execution of a specific timer. Returns true if the timer was running and is now stopped.

startTimer

The function “startTimer” on page 953 restarts the execution of a specific timer. Returns true if the timer was stopped and is now restarted.

isTimer

The function “isTimer” on page 954 returns TRUE if \( \text{id} \) is a valid action index and it is a timer.

getTimerName

The function “getTimerName” on page 954 returns a string containing the name of the timer (or NULL if not a timer).

getTimerInterval

The function “getTimerInterval” on page 954 returns the number of seconds between each execution of the timer.

getTimerID

The function “getTimerID” on page 954 returns the timer \( \text{id} \) as an integer.

getTimerRunning

The function “getTimerRunning” on page 955 returns TRUE if \( \text{id} \) is a timer and it is running.
Triggers

The documentation for triggers is updated in version 9.6.1.6 to include this statement: Rational DOORS Web Access does not support triggers. This condition applies to all previous versions also.
Chapter 3

New in DXL for Rational DOORS 9.6

This chapter describes features and documentation that are new in Rational DOORS 9.6:

• Operating system interface: getMemoryUsage
• Mini database explorer
• Modules: downgrade, downgradeShare
• History constants
• Dialog box functions: common element operations
• Display control
  • Columns
  • Layout DXL: setRefreshDelta
• HTTP Server

Operating system interface

getMemoryUsage

This perm “getMemoryUsage” on page 178 returns the Rational DOORS client memory usage in MB.
The perm was added in Rational DOORS 9.3.0.7, but the documentation is new in 9.6.0.1.

Mini database explorer

This perm “fnMiniExplorer” on page 198 creates a miniature database explorer window that shows a tree view in which you can navigate through the hierarchy of the Rational DOORS database and select an item.

This information is added to the documentation in Rational DOORS 9.6.0.1. The function is available in previous releases.
Modules

downgrade

This information is added to the documentation for the existing perm “downgrade” on page 299 in Rational DOORS 9.6.0.1:

If there are unsaved changes to the module, then the user is prompted to save the changes. Alternatively, the save perm can be used prior to downgrade, so that any changes to the module are preserved.

downgradeShare

This information is added to the documentation for the existing perm “downgradeShare” on page 299 in Rational DOORS 9.6.0.1:

If there are unsaved changes to the module, then the user is prompted to save the changes. Alternatively, the save perm can be used prior to downgrade, so that any changes to the module are preserved.

History

Constants (history type)

The following constants are added to the list of “Constants (history type)” on page 331:

• const HistoryType moveObject
• const HistoryType synchronizeModule
• const HistoryType commentModule
• const HistoryType commentObject

This information is added to the documentation in Rational DOORS 9.6.0.1. The functions are available in previous releases.
Dialog box functions: common element operations

setTextChangeCB

This perm “setTextChangeCB” on page 510 sets the text change callback for field, richField, text, richText DBEs where the callback is of the form `void callbackFn(DBE)`. When the callback function is invoked on a text change, DBE will be the handle of the edit control DBE, which can be field, richField, text or richText.

This is a new function in Rational DOORS 9.6.

toolBarEditGetString

This perm “toolBarEditGetString” on page 511 gets the contents of the edit control hosted on the toolbar with DBE handle `tb`, where `index` identifies the edit control on this toolbar by the index of the edit control.

This is a new function in Rational DOORS 9.6.

Display control: columns

link(get)

This perm “link(get)” on page 694 returns true if column `c` is a link indicator column.

This is a new function in Rational DOORS 9.6.

link(set)

This perm “link(set)” on page 694 makes column `c` a link indicator column.

This is a new function in Rational DOORS 9.6.

changebar(get)

This perm “changebar(get)” on page 694 returns true if column `c` is a change bar column.

This is a new function in Rational DOORS 9.6.

changebar(set)

This perm “changebar(set)” on page 694 makes column `c` a change bar column.
This is a new function in Rational DOORS 9.6.

---

**currentColumn(get)**

This perm “currentColumn(get)” on page 696 gets the current column for this DXL context. If the DXL is not layout DXL then this will return NULL.

This information is added to the documentation in Rational DOORS 9.6. The function is available in previous releases.

---

**Display control: Layout DXL**

---

**setRefreshDelta**

The function “setRefreshDelta” on page 702 is a new Layout DXL function in Rational DOORS 9.6. The documentation for this function is added in 9.6.1.

---

**HTTP Server**

The new section “HTTP Server” on page 940 defines functions for making HTTP requests to a URL.

This information is added to the documentation in Rational DOORS 9.6. The functions are available in the most recent previous releases.
Chapter 4
New in DXL for Rational DOORS 9.5

This chapter describes features and documentation that are new in Rational DOORS 9.5:

- Embedded OLE objects and the OLE clipboard
- OSLC Link Discovery
- Database properties
- Rational Directory Server

Embedded OLE objects and the OLE clipboard

oleInsert

Declaration

bool oleInsert(Object o,[attrRef],string fileName,[bool insertAsIcon])

where the optional parameter attrRef is in the following format: (Object o).(string attrName)

Operation

Embeds the file fileName as an OLE object in the Rational DOORS formal object o in a text attribute. If the optional parameter attrRef is specified, then the OLE object is embedded in the user-defined text attribute. If no parameter is specified, then the OLE object is embedded in the system Object Text attribute.

If the optional parameter insertAsIcon is specified, then if true, the OLE object is displayed as an icon, else it is displayed as content. If no parameter is specified, then the default is to display the OLE object as content.

The function returns true on successful insertion of the OLE object. Otherwise, it returns false.

An OLE package is created if a file has no associated applications that support OLE. OLE packages even allow executable files to be embedded into documents. It is then possible to execute such a file from within the document.

Example

/*
this code segment embeds an existing word document into the current formal object
*/

string docName = "c:\\docs\\details.doc"
Object obj = current
if (oleInsert(obj, obj."my_text", docName)){
    print "Successfully embedded document\n"
} else {
    print "Problem trying to embed document\n"
}

OSLC Link Discovery

When OSLC (external) links are discovered the results are stored in DOORS in a database-wide cache so that future sessions that open modules with those links open faster. When a user opens a module, the cache is checked first for any external links. If the data in the cache has not yet expired then the cached external links are shown; else, a new query is executed to discover any OSLC (external) links and the cache is then updated with the results. The cache has a default expiry time of 5 minutes after which the external links are considered to be out of date. This expiry time can be modified.

getCachedExternalLinkLifeTime

Declaration

int getCachedExternalLinkLifeTime()

Operation

Returns the life time (expiry time) of the cached external links in seconds.

setCachedExternalLinkLifeTime

Declaration

string setCachedExternalLinkLifeTime(int lifetime)

Operation

Sets the life time (expiry time) of the cached external links to lifetime seconds.

If the value lifetime is zero then this will disable link discovery.

Returns an error if the user does not have the manage database privilege; otherwise, returns null.
Database properties

getReconfirmPasswordRequired

**Declaration**

bool getReconfirmPasswordRequired()

**Operation**

Returns `true` if a reconfirmation password is required after a specified timeout period; otherwise, returns `false`.

setReconfirmPasswordRequired

**Declaration**

void setReconfirmPasswordRequired(bool required)

**Operation**

Sets whether a reconfirmation password is required after a specified timeout period, depending on the value of `required`. This perm only operates if the current user has the Manage Database privilege.

getReconfirmPasswordTimeout

**Declaration**

int getReconfirmPasswordTimeout()

**Operation**

Returns the timeout period (in minutes) before the reconfirmation password dialog appears.

setReconfirmPasswordTimeout

**Declaration**

void setReconfirmPasswordTimeout(int timeout)

**Operation**

Sets the timeout period to `timeout` minutes before the reconfirmation password dialog appears. This perm only operates if the current user has the Manage Database privilege.
getRequireLettersInPassword

Declaration
bool getRequireLettersInPassword()

Operation
Returns true if a password is required to contain at least one alphabetic character; otherwise, returns false.

setRequireLettersInPassword

Declaration
string setRequireLettersInPassword(bool required)

Operation
If required is true, then a password is required to contain at least one alphabetic character.
This perm only operates if the current user has the Manage Database privilege, otherwise an error message is returned.

getRequireNumberInPassword

Declaration
bool getRequireNumberInPassword()

Operation
Returns true if a password is required to contain at least one number; otherwise, returns false.

setRequireNumberInPassword

Declaration
string setRequireNumberInPassword(bool required)

Operation
If required is true, a password is required to contain at least one number.
This perm only operates if the current user has the Manage Database privilege, otherwise an error message is returned.

getRequireSymbolInPassword

Declaration
bool getRequireSymbolInPassword()
Operation
Returns true if a password is required to contain at least one non-alphanumeric character; otherwise, returns false.

setRequireSymbolInPassword

Declaration
string setRequireSymbolInPassword(bool required)

Operation
If required is true, a password is required to contain at least one non-alphanumeric character.
This perm only operates if the current user has the Manage Database privilege, otherwise an error message is returned.

getMinPasswordGeneration

Declaration
int getMinPasswordGeneration()

Operation
Returns the minimum number of password generations before a password can be reused.

setMinPasswordGeneration

Declaration
string setMinPasswordGeneration(int num)

Operation
Sets the minimum number of password generations before a password can be reused to num. The minimum number cannot exceed the in-built maximum limit of 12 generations before a password can be reused.
This perm only operates if the current user has the Manage Database privilege, otherwise an error message is returned.

getMaxPasswordGenerationLimit

Declaration
int getMaxPasswordGenerationLimit()

Operation
Returns the in-built maximum limit of password generations before a password can be reused. This maximum limit is set to 12.
getMinPasswordAgeInDays

Declaration
int getMinPasswordAgeInDays()

Operation
Returns the minimum number of days before a password can be reused.

setMinMaxPasswordAgeInDays

Declaration
string setMinMaxPasswordAgeInDays(int days)

Operation
Sets the minimum number of days before a password can be reused to days. The minimum number cannot exceed the in-built maximum limit of 180 days before a password can be reused.

This perm only operates if the current user has the Manage Database privilege, otherwise an error message is returned.

getMaxPasswordAgeLimit

Declaration
int getMaxPasswordAgeLimit()

Operation
Returns the in-built maximum limit of days before a password can be reused. This maximum limit is set to 180 days.

Rational Directory Server

getTDPortNo

Declaration
int getTDPortNo()

Operation
Returns the Rational Directory Server port number.
setTDPortNo

Declaration

```
string setTDPortNo(int i)
```

Operation

Sets the Rational Directory Server port number.

Returns an error string if the current user is not the administrator.
Chapter 5
New in DXL for Rational DOORS 9.4

This chapter describes features that are new in Rational DOORS 9.4:

- Attribute definitions
- Attribute types
- Rich text strings

Attribute definitions

Attribute definition properties

Properties are defined for use with the . (dot) operator and an attribute definition handle to extract information from an attribute definition, as shown in the following syntax:

(AttrDef ad).property

The following property is now supported:

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>uri</td>
<td>The URI of an attribute definition.</td>
</tr>
</tbody>
</table>

create(attribute definition)

Syntax

AttrDef create([module|object]
                [property value]...
                [(default defVal)]
                attribute(string attrName))

Operation

Creates a new attribute definition called attrName from the call to attribute, which is the only argument that must be passed to create. The optional arguments modify create, by specifying the value of attribute properties. The arguments can be concatenated together to form valid attribute creation statements.

The keywords module and object specify that the attribute definition that is being created applies to modules or objects, respectively.
The default property specifies the default value for the attribute definition that is being created as \texttt{defVal}. This property should always be specified within parenthesis to avoid parsing problems. The value must be given as a string, even if the underlying type is different. Rational DOORS converts the value automatically.

As required, you can specify other properties. The defaults are the same as the Rational DOORS user interface. The following property is now supported:

<table>
<thead>
<tr>
<th>String property</th>
<th>Specifies</th>
</tr>
</thead>
<tbody>
<tr>
<td>uri</td>
<td>The URI of an attribute definition.</td>
</tr>
</tbody>
</table>

**modify(attribute definition)**

**Declaration**

\begin{verbatim}
AttrDef modify(AttrDef old,
              [setproperty value,]
              AttrDef new)
\end{verbatim}

**Operation**

Modifies an existing attribute definition by passing it a new attribute definition. The optional second argument enables you to set a single property. The following property is now supported:

<table>
<thead>
<tr>
<th>String property</th>
<th>Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>uri</td>
<td>The URI of an attribute definition.</td>
</tr>
</tbody>
</table>

**Example**

AttrDef ad = create object type "Integer" attribute "cost"
\begin{verbatim}
ad = modify(ad, object type "Integer" attribute "Costing")
ad = modify(ad, setHistory, true)
ad = modify(ad, setDefault, "123")
ad = modify(ad, setURI, "http://www.webaddress.com")
\end{verbatim}

**Attribute types**

**setURI**

**Declaration**

\begin{verbatim}
AttrType setURI(AttrType at, string URI, string &errMess)
\end{verbatim}
AttrType setURI(AttrType at, string name, string URI, string &errMess)
AttrType setURI(AttrType at, int index, string URI, string &errMess)

**Operation**
Sets the URI for the specified attribute type. Returns a modified attribute type. If there is an error, the message is returned in the final string parameter. The URI can be set for a specified enumeration value or enumeration index.

**Example**
AttrType at
string errorMsg

string index[] = { "first", "second", "third" }

at = setURI(at, "http://www.webaddress.com", errorMsg)
at = setURI(at, index[0], "http://www.webaddress.com", errorMsg)

---

**getURI**

**Declaration**

string uri(AttrType at)
string uri(AttrType at, string name)
string uri(AttrType at, int index)

**Operation**
Gets the URI for the specified attribute type or for a named enumeration value or for a enumeration index.

---

**Rich text strings**

---

**applyTextFormattingToParagraph**

**Declaration**

string applyTextFormattingToParagraph(string s, bool addBullets, int indentLevel, int paraNumber, [int firstIndent])

**Operation**
Applies bullet and/or indent style to the given text, overwriting any existing bullets/indents.

- If `addBullets` is `true`, adds bullet style.
- If `indentLevel` is nonzero, adds indenting to the value of `indentLevel`. The units for `indentLevel` are twips = twentieths of a point.
• If \textit{paraNumber} is zero, the formatting is applied to all the text. Otherwise it is only applied to the specified paragraph number.

• If the optional parameter \textit{firstIndent} is specified, then this sets the first line indent. If the value is negative then this sets a hanging indent. The units are in points.

The input string \textit{s} must be rich text. For example, from \textit{string s = richText o."Object Text"}.

Returns a rich text string which describes the text with the formatting applied.

\textbf{Example}

Object o = current

string s = o."Object text"

o."Object text" = richText (applyTextFormattingToParagraph(richText s, true, 0, 0))

Adds bullet style to all of the current object’s text.
Chapter 6
New in DXL for Rational DOORS 9.3

This chapter describes features that are new in Rational DOORS 9.3:

• Converting a symbol character to Unicode
• Dialog box functions
• Operations on type string
• Embedded OLE objects and the OLE clipboard
• OLE information functions
• Discussions
• RIF ID
• Rational DOORS URLs
• Filters
• Compound Filters
• Localizing DXL
• Finding links
• Links

Converting a symbol character to Unicode

symbolToUnicode

Declaration
char symbolToUnicode(char symbolChar, bool convertAllSymbols)

Operation
Converts a symbol character to its Unicode equivalent. If convertAllSymbols is false, only symbols with the Times New Roman font equivalents are converted.
Dialog box functions

addAcceleratorKey

Declaration
void addAcceleratorKey(DB db, void dxlCallback(), char accelerator, int modifierKeyFlags)

Operation
Adds an accelerator key accelerator to the dialog db with the callback function dxlCallback() and the passed-in modifierKeyFlags. modifierKeyFlags is used in conjunction with the accelerator parameter to change which key should be pressed with the accelerator key. Possible values for it are modKeyNone, modKeyCtrl, modKeyShift and null.

The specified DXL callback fn dxlCallback() executes for the specified keystroke combination being pressed when the DXL dialog box db is active.

Only call this perm after the dialog box db has been realized, otherwise a DXL run-time error will occur.

Example
void fn()
{
    print "callback fires\n"
}
DB db = create("testDialog", styleStandard)
realize db

    // The callback fn() will be executed on pressing Shift+F7 when the dialog db is active.
    addAcceleratorKey(db, fn, keyF7, modKeyShift)

Operations on type string

unicodeString

Declaration
string unicodeString(RTF_string__ str, bool convertAllSymbols, bool returnAsPlainText)
Operation
Returns the value of the specified rich text string as RTF or plain text. If the attribute contains characters in Symbol font, these characters are converted to the Unicode equivalents.

If convertAllSymbols is true, all symbol character are converted. If false, only Unicode characters that have a good chance of being displayed are used. See the symbolToUnicode perm for a description of which characters are converted.

The value is returned as plain text if returnAsPlainText is true. Otherwise the value is returned as RTF.

escape

Declaration
string escape(string str, char escapeChar, string escapeChars)

Operation
Escapes all the characters in str which are in escapeChars, with the escapeChar character. This also escapes escapeChar itself.

Example
escape("hello world", '/', "l") returns "he/l/lo wor/ld"
escape("hello world #1", '#', "lh") returns "#he#l#lo wor#ld ##1"

stripPath

Declaration
string stripPath(string path, bool isEscaped)

Operation
Removes the path part from path, using forward slash as the path separator.
If isEscaped is true, the slash character can be used as a literal character rather than a path separator by preceding the character with a backslash.

Example
stripPath("abc/def/ghi", b) returns "ghi", where b is true or false.
stripPath("abc/def\/ghi", true) returns "def/ghi"
Embedded OLE objects and the OLE clipboard

olePasteSpecial

Declaration

string olePasteSpecial(string attrRef, bool displayAsIcon)

Operation

Copies an OLE object from the clipboard and appends it to attrRef. The boolean displayAsIcon, when set to true will display the OLE object as an icon in the object. Returns null on success and displays an error message on failure.

Example

Object o = current
olePasteSpecial(o."object text", false)

OLE information functions

oleSetHeightandWidth

Declaration

oleSetHeightandWidth(string attrRef, int height, int width, int index)

Operation

Sets the height and width of the OLE object within attrRef at the specified index.

Example

Object o = current Object
oleSetHeightandWidth(o."Object Text", 150, 150, 1)
## Discussions

### isDiscussionColumn

**Declaration**

```c
bool isDiscussionColumn(Column c)
```

**Operation**

Returns true if the column is a discussion column, otherwise false.

### setDiscussionColumn

**Declaration**

```c
void setDiscussionColumn(Column c, string s)
```

**Operation**

Sets the filter on the discussion column based on the supplied discussion DXL filename.

**Example**

```c
Column c
for c in current Module do
{
    if (isDiscussionColumn(c))
    {
        string s = dxlFilename(c)
        if (s != null)
        {
            Module m = edit("/TestDiscussions ", true)
            //Open a module, with some discussions in it.
            if (m != null)
            {
                Column cNew = insert(column 3)
                title(cNew, "My copy Discussion")
                string home = getenv("HOME")
                string fullPath = home "\\" s ""
                string contents = readFile(fullPath)
            }
        }
    }
}
```
// Call dxl PERM on that column before setting the discussion column. The discussion column is also a modified version of LAYOUT dxl.

dxl(cNew, contents)
setDiscussionColumn(cNew, s)
width(cNew, 100)
refresh(m, false)


canModifyDiscussions

Declaration

bool canModifyDiscussions({Module m| Item i| string s}[, {User |string}])

Operation

Returns true if a given user or named user (current user if the parameter is not supplied) is allowed to create a discussion or a comment on a discussion for the given module, item or named module. The use of item is intended for use when the Item represents a module.

canEveryoneModifyDiscussions

Declaration

bool canEveryoneModifyDiscussions({Module m| Item i})

Operation

Returns true if the discussions access list for the given module or item contains the special "Everyone" group.

addUser

Declaration

void addUser(Item i, {User u| string s})

Operation

Adds the user or named user to the Discussion Access List for an Item. The updated list is not saved in the database until saveDiscussionAccessList is called.
addGroup

Declaration

```c
void addGroup(Item i, {Group g| string s})
```

Operation

Adds the group or named group to the Discussion Access List for an Item. The updated list is not saved in the database until `saveDiscussionAccessList` is called.

removeUser

Declaration

```c
void RemoveUser(Item i, {User u| string s})
```

Operation

Remove the user or named user from the Discussion Access List for an Item. The updated list is not saved in the database until `saveDiscussionAccessList` is called.

removeGroup

Declaration

```c
void removeGroup(Item i, {Group g| string s})
```

Operation

Remove the group or named group from the Discussion Access List for an Item. The updated list is not saved in the database until `saveDiscussionAccessList` is called.

saveDiscussionAccessList

Declaration

```c
string saveDiscussionAccessList(Item i)
```

Operation

This perm saves the discussion access list for the given item to the database. This perm is only successful for an administrator or a user with manage database privileges. If the call is successful, a null value will be returned, otherwise a string with an error message will be returned.
RIF ID

getRifID

Declaration

string getRifID(Object o)

Operation

Returns a string with the RIF ID for object o. If the object does not have a RIF ID, an empty string is returned.

getObjectByRifID

Declaration

Object getObjectByRifID(Module m, string s)

Operation

Returns the object within module m with a RIF ID of s. If the module does not contain an object with the input RIF ID, null is returned.

Rational DOORS URLs

getResourceURL

Declaration

string getResourceURL(Module | Object | Database__ | ModuleVersion | ModName__ | Folder | Project | Item)

Operation

Returns the resource URL of the passed in item.

getResourceURLConfigOptions

Declaration

void getResourceURLConfigOptions(string &dwaProtocol, string &dwaHost, int &dwaPort)
Operation

Gets the `dwaProtocol`, `dwaHost`, and `dwaPort` DBAdmin options configured for this database. The `dwaProtocol`, `dwaHost`, and `dwaPort` parameters contain the values upon return.

decodeResourceURL

Declaration

```cpp
string decodeResourceURL(string resourceURL, string &protocol, string& dbHost, int& dbPort, string& repositoryId, string& dbId, string& dbName, string& dbId, Item&, ModuleVersion&, string& viewName, int& objectAbsno)
```

Operation

Breaks down a passed-in resource URL into its constituent parts and passes back the information as may be applicable into the reference parameters.

Returns null on success, error message on failure.

Filters

getSimpleFilterType_

Declaration

```cpp
int getSimpleFilterType_(Filter)
```

Operation

Returns the type of the simple filter; attribute, link, object, or column. Please note that the returned value corresponds to the index of the appropriate tab page on the filter dialog. If the specified filter is not a simple filter, -1 is returned.

getAttributeFilterSettings_

Declaration

```cpp
bool getAttributeFilterSettings_(Module, Filter, string& attributeName, int& comparisonType, string& comparisonValue, bool& matchCase, bool& useRegexp)
```
Operation

Gets details of the specified attribute filter in the return parameters. The function returns false if the filter is not a valid attribute filter.

The comparisonType parameter returns the internal index of the comparison. This is different to the index that is used in the associated combo box on the filter dialog. The translation is performed by the DXL code.

getLinkFilterSettings_

Declaration

bool getLinkFilterSettings_(Module,
    Filter,
    bool& mustHave,
    int& linkType,
    string& linkModuleName)

Operation

Gets details of the specified link filter in the return parameters. The function returns false if the filter is not a valid link filter.

The linkType parameter returns a value that maps directly to the appropriate combo box.

The linkModuleName parameter returns an asterisk if links are allowed through any module, or the module name.

getObjectFilterSettings_

Declaration

bool getObjectFilterSettings_(Module,
    Filter,
    int& objectFilterType)

Operation

Gets details of the specified object filter in the return parameter. The function returns false if the filter is not a valid object filter.

The objectFilterType parameter returns a value that maps directly to the radio group on the dialog.

columnFilterSettings_

Declaration

bool getColumnFilterSettings_(Module,
    Filter,
stringValue,  
bool& matchCase,  
bool& useRegExp)

Operation

Gets details of the specified column filter in the return parameters. The function returns false if the filter is not a valid column filter.

Compound Filters

These perms can be used to decompose compound filters into their component parts for analysis, and potential modification or replacement.

See examples in “Compound filters” on page 656.

getCompoundFilterType_

Declaration

int getCompoundFilterType_(Filter)

Operation

Returns an integer value indicating the type of the specified filter.

It returns one of the following new DXL constant values for compound filter types:

int filterTypeAnd
int filterTypeOr
int filterTypeNot

It returns -1 for a simple filter. The test for a negative value suffices to indicate that the filter is not compound, as the new constants are all positive values.

If no filter is supplied, a run-time DXL error is generated.

getComponentFilter_

Declaration

Filter getComponentFilter_(Filter, int index)

Operation

Returns an integer value indicating the type of the specified filter.
It returns one of the following new DXL constant values for compound filter types:

```dxl
int filterTypeAnd
int filterTypeOr
int filterTypeNot
```

This perm returns a component filter that is part of the supplied compound filter. If the compound filter is of type `filterTypeNot`, the index must be zero, or the perm returns `null`. If the compound filter is of type `filterTypeOr` or `filterTypeAnd`, an index of 0 or 1 returns the first or second sub-filter, and any other index value returns `null`.

If the supplied filter is not a compound filter, the perm returns `null`.

If no filter is supplied, a run-time DXL error is generated.

---

**Localizing DXL**

Rational DOORS uses ICU resource bundles for accessing translated strings. DXL perms are available to access ICU resource bundles containing translated strings for customized DXL. For information about creating ICU resource bundles, see [http://userguide.icu-project.org/locale/localizing](http://userguide.icu-project.org/locale/localizing).

Put the language resource files in a directory whose name is taken as the *bundle name*, under `$DOORSHOME/language`, for example `$DOORSHOME/language/myResource/de_DE.res`. There are two bundles already shipped with Rational DOORS, `core` and `DXL`.

---

**LS_**

**Declaration**

```dxl
string LS_(string key, string fallback, string bundle)
```

**Operation**

Returns the string from resource bundle that is identified by key. If the string identified by key is not found in the resource bundle, the fallback string is returned.

**Example**

de.txt file contains;

```txt
de {
    Key1{"Ausgehend"}
    Key2{"Ausgehende Links"}
    Key3{"Normalansicht"}
    Key4{"Klartext"}
}
```
From the command line, generate a resource bundle, for example genrb de.txt, and copy the resource bundle to $DOORSHOME/language/myResource/, where myResource is the name of your resource bundle. The localized strings can then be accessed using the LS_perm, for example in the DXL editor, type:

```dxl
print LS_("Key1", "Ausgehend not found", "myResource") "\n"
print LS_("Key2", "Ausgehende Links not found", "myResource") "\n"
print LS_("Key3", "Normalansicht not found", "myResource") "\n"
print LS_("Key4", "Klartext not found", "myResource") "\n"
```

The output is:

Ausgehend
Ausgehende Links
Normalansicht
Klartext

---

**Finding links**

---

**for each incoming link**

---

**Syntax**

```dxl
for (LinkRef) in Object tgtObject<-string
    linkModuleName) do {
    ...
}
```

where:

- `LinkRef` is a variable of type Link or LinkRef
- `tgtObject` is a variable of type Object
- `linkModuleName` is a string variable

**Operation**

Assigns the variable `LinkRef` to be each successive incoming link arriving at object `tgtObject` via link module named `linkModuleName`. The string `linkModuleName` can be a specific link module name, or the string "*" meaning any link module.

Iterates through all incoming link references including those from baselines and soft-deleted modules.

**Note:** This loop only assigns to `LinkRef` incoming link values for which the source object is loaded; unloaded links are not detected.
Example

LinkRef l

for (l) in current Object<"*" do {
    string user = l."Created By"
    print user "\n"
}

Syntax

for (srcModName) in Object tgtObject<string linkModName) do {
    ...
}

where:

srcModName is a string variable
tgtObject is a variable of type Object
linkModName is a string variable

Operation

Assigns the variable srcModName to be the unqualified name of the source module of each successive incoming link arriving at object tgtObject via link module named linkModuleName. The string linkModuleName can be a specific link module name, or the string "*" meaning any link module.

Includes links from baselines and soft-deleted modules, returning the name of the source module (without baseline version numbers).

Note: This loop assigns to modName values for all incoming links, whether the source is loaded or not. This can be used to pre-load all incoming link sources before using the for all incoming links loop.

Example

This example prints the unqualified name of all the source modules for incoming links to the current object:

Object o = current
string srcModName
for (srcModName) in o<"*" do print srcModName "\n"
for each source reference

Syntax

for (srcModRef) in Object tgtObject<-string
    linkModName) do {
    ...
}

where:

srcModRef is a variable of type ModName_
tgtObject is a variable of type Object
linkModName is a string variable

Operation

Assigns the variable srcModRef to be the reference of the source module of each successive incoming link arriving at object tgtObject via link module named linkModuleName. The string linkModuleName can be a specific link module name, or the string "*" meaning any link module.

Includes links from baselines and soft-deleted modules.

Note: This loop assigns to modName values for all incoming links, whether the source is loaded or not. This can be used to pre-load all incoming link sources before using the for all incoming links loop.

Example

ModName_ srcModRef
for (srcModRef) in o<-"*" do
    read(fullName(srcModRef), false)

Links

getlegacyURL

Declaration

string getLegacyURL(object o)

Operation

This perm returns the legacy Rational DOORS URL. The legacy URL contains the protocol as "doors". This URL can then be decoded using decodeURL.
Example
ModuleVersion mv
int objectAbsno
Item i
string dbHost = null
int dbPort
string dbName
string dbID = null

string objUrl = getURL(current Object)

string legacyUrl
string errorMsg
errorMsg = getLegacyURL(objUrl, legacyUrl)
if(!null errorMsg)
{
    print errorMsg "\n"
}
else
{
    errorMsg = decodeURL(legacyUrl, dbHost, dbPort, dbName, dbID, i, mv, objectAbsno)
}
if(!null errorMsg)
{
    print errorMsg "\n"
}
else
{
    print "Original URL - " objUrl "\nDB Host - " dbHost "\n" print "DB Port - " dbPort "\nDB Name - " dbName "\nDB Id - " dbId "\nAbsolute Number - " objectAbsno "\n"
Chapter 7

New in DXL for Rational DOORS 9.2

This chapter describes features that are new in Rational DOORS 9.2:

- Additional authentication
- Dialog box updates
- New constants
- Partitions updates
- Requirements Interchange Format (RIF)

Additional authentication

getAdditionalAuthenticationEnabled

Declaration

bool getAdditionalAuthenticationEnabled()

Operation

Returns true if enhanced security users need to perform additional authentication during login. Only relevant when authentication is being controlled via RDS.

getAdditionalAuthenticationPrompt

Declaration

string getAdditionalAuthenticationPrompt()

Operation

Returns the label under which additional authentication is requested, if enhanced security is enabled, for example the label for the second “password” field. Only relevant when authentication is being controlled via RDS.

getsystemLoginConformityRequired

Declaration

bool getsystemLoginConformityRequired()
Operation

Returns true if enhanced security users have their system login verified when logging in. Only relevant when authentication is being controlled via RDS.

getCommandLinePasswordDisabled

Declaration

bool getCommandLinePasswordDisabled()

Operation

Return true if the -P command line password argument is disabled by default.

setCommandLinePasswordDisabled

Declaration

string setCommandLinePasswordDisabled(bool)

Operation

Sets whether the -P command line password argument is disabled by default. Supplying true disables the option by default.

Dialog box updates

toolBarComboGetEditBoxSelection

Declaration

string toolBarComboGetEditBoxSelection(DBE toolbar, int index)

Operation

Returns the selected text from the editable combo box in toolbar where index is the combo box index.

toolBarComboCutCopySelectedText

Declaration

void toolBarComboCutCopySelectedText(DBE toolbar, int index, bool cut)
Operation
Cuts, or copies, the selected text in the editable combo box in toolbar at location index. If cut is true, the selected text is cut to the clipboard. Otherwise, it is copied.

**toolBarComboPasteText**

Declaration
void toolBarComboPasteText(DBE toolbar, int index)

Operation
Pastes text from the clipboard into the combo box located at index in toolbar. Replaces selected text if there is any.

**hasFocus**

Declaration
bool hasFocus(DBE toolbar)

Operation
Returns true if the supplied toolbar DBE contains an element that currently has the keyboard focus. Otherwise, returns false.

**setDXLWindowAsParent**

Declaration
void setDXLWindowAsParent(DB dialog)

Operation
Sets the DXL interaction window to be the parent of dialog. If there is no DXL interaction window, the parent is set to null.

**New constants**

**mayUseCommandLinePassword**

Declaration
bool mayUseCommandLinePassword
**Operation**

Boolean property of a *User*. When command line passwords are disabled by default, this returns *true* if they have been enabled for the given *User*. Otherwise, returns *false*.

---

**additionalAuthenticationRequired**

**Declaration**

```c
bool additionalAuthenticationRequired
```

**Operation**

Boolean property of a *User*. Returns *true* if the *User* needs to perform additional authentication during login. Only relevant when authentication is performed via RDS.

---

**iconAuthenticatingUser**

**Declaration**

```c
Icon iconAuthenticatingUser
```

**Operation**

The icon used to represent a user required to perform additional authentication during login.

---

**Partitions updates**

**addAwayModule**

**Declaration**

```c
string addAwayModule(PartitionDefinition pd, string modName[, string partName])
```

**Operation**

Used to add a formal module to a partition in the away database.

The new, optional parameter can be used to specify the partition name where it may vary from the definition name.

---

**addAwayLinkModule**

**Declaration**

```c
string addAwayLinkModule(PartitionDefinition pd, string modName[, string partName])
```
Operation

Used to add a link module to a partition in the away database.

The new, optional parameter can be used to specify the partition name where it may vary from the definition name.

Requirements Interchange Format (RIF)

exportType

Declaration

void initRIFExport(ExportType)

Operation

Sets the export type to be either RIF or ReqIF. Call this before using the exportPackage method.

example

initRIFExport(exportRIF_1_2) // Sets the export to be RIF
initRIFExport(exportReqIF) // Sets the export to be ReqIF

exportPackage

Declaration

string exportPackage(RifDefinition def, Stream RifFile, DB parent, bool& cancel)

Operation

Exports def to the XML file identified by RifFile. The stream must be have been opened for writing using “write (filename, CP_UTF8)”. If parent is null then a non-interactive operation is performed. Otherwise, progress bars will be displayed.

If an interactive export is performed, and is cancelled by the user, cancel will be set to true.

importRifFile

Declaration

string importRifFile(string RifFilename, Folder parent, string targetName, string targetDesc, string RifDefName, string RifDefDescription, DB parent)

Operation

Performs a non-interactive import of RifFilename, placing the imported modules in a new folder in the specified parent. The new folder name and description are specified by targetName and targetDesc.
rifMerge

Declaration

string rifMerge(RifImport mrgObj, DB parent)

Operation

Performs a non-interactive merge using the information in mrgObj.

RifDefinition

A RifDefinition is the object in which a package to be exported in RIF format is defined.

Properties are defined for use with the . (dot) operator and a RifDefinition handle to extract information from a definition, as shown in the following syntax:

variable.property

where:

- variable is a variable of type RifDefinition.
- property is one of the following properties.

The following tables list the RifDefinition properties and the information they extract or specify:

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the definition.</td>
</tr>
<tr>
<td>description</td>
<td>The description of the definition.</td>
</tr>
<tr>
<td>rifDefinitionIdentifier</td>
<td>The unique ID of the RIF definition (this is shared between databases, unlike the name and description).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>boolean property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>createdLocally</td>
<td>Returns true if the definition was created in the local database, as opposed to being imported.</td>
</tr>
<tr>
<td>canModify</td>
<td>Returns true if the correct user can modify the definition.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>project</td>
<td>The project which contains the definition.</td>
</tr>
</tbody>
</table>
A **RifModuleDefinition** is an object which contains the details of how a module should be exported, as part of a RIF package.

Properties are defined for use with the . (dot) operator and **RifModuleDefinition** handle to extract information from a definition record, as shown in the following syntax:

```plaintext
variable.property
```

where:

- `variable` is a variable of type **RifModuleDefinition**.
- `property` is one of the properties below.

The following tables list the **RifModuleDefinition** properties and the information they extract or specify:

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataConfigView</td>
<td>The name of the view used to define which data in the module will be included in the RIF export.</td>
</tr>
<tr>
<td>ddcView</td>
<td>The name of the view used to define what data can be edited when the exported RIF package is imported into another database.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bool property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>createdLocally</td>
<td>Whether the module was added to the RifDefinition in the current database or not.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ModuleVersion property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleVersion</td>
<td>The ModuleVersion reference for the given RifModuleDefinition.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ddcmode property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddcMode</td>
<td>The type of access control used to define whether the module, or its contents, will be editable in each database once it has been exported.</td>
</tr>
</tbody>
</table>
DdcMode constants

DdcMode constants define the type of access control used to define whether a module, or its contents, will be editable in each of the local and target database once the export has taken place. The following table details the possible values, and their meanings.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddcNone</td>
<td>Module will be editable in both source and target databases.</td>
</tr>
<tr>
<td>ddcReadOnly</td>
<td>Module will be editable in only the source database.</td>
</tr>
<tr>
<td>ddcByObject</td>
<td>Selected objects in the module will be made read-only in the source database.</td>
</tr>
<tr>
<td>ddcByAttribute</td>
<td>Selected attributes in the module will be made read-only in the source database.</td>
</tr>
<tr>
<td>ddcFullModule</td>
<td>Module will not be editable.</td>
</tr>
</tbody>
</table>

RifImport

A RifImport is an object which contains information on a RIF import. These are created by import operations, and are persisted in a list in the stored RifDefinition.

Properties are defined for use with the . (dot) operator and a RifImport handle to extract information from, or specify information in an import record, as shown in the following syntax:

```
variable.property
```

where:

- `variable` is a variable of type RifImport.
- `property` is one of the properties.

The following tables list the RifImport properties and the information they extract or specify:

<table>
<thead>
<tr>
<th>bool property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>mergeStarted</td>
<td>Returns true when a merge operation is started.</td>
</tr>
<tr>
<td>mergeCompleted</td>
<td>Returns true when the merge has been completed.</td>
</tr>
<tr>
<td>mergeRequired</td>
<td>Returns true when an import is a valid candidate for merging.</td>
</tr>
<tr>
<td>mergeDisabled</td>
<td>Returns true if the merge has been disabled due to lock removal.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>importedBy</td>
<td>Returns the user who performed the import.</td>
</tr>
</tbody>
</table>
for RifDefinition in Project

Syntax

\[ \text{for rifDef in proj do} \{ \]
\[ \text{\quad ...} \]
\[ \text{\}} \]

Operation

Assigns \textit{rifDef} to be each successive \texttt{RifDefinition} in Project \textit{proj}.

for RifModuleDefinition in RifDefinition

Syntax

\[ \text{for rifModDef in rifDef so} \{ \]
\[ \text{\quad ...} \]
\[ \text{\} } \]
Operation

Assigns `rifModDef` to be each successive `RifModuleDefinition` in `RifDefinition rifDef`.

```
for RifImport in RifDefinition
```

Syntax

```
for rifImp in rifDef do {
...
}
```

Operation

Assigns `rifImp` to be each successive `rifImport` in `RifDefinition rifDef`.

Examples

The following example dumps all information about all RIF definitions in the current project to the screen. It then conditional exports one of the packages.

```
RifDefinition rd
RifModuleDefinition rmd
Stream stm = write ("C:\Public\rifExport.xml", CP_UTF8)
string s = ""
bool b
Project p = current
Project p2
ModuleVersion mv
DB myDB = null
DdcMode ddcm

for rd in p do {

    print rd.name "\n"
    print rd.description "\n"
    print rd.rifDefinitionIdentifier "\n"

    if (rd.createdLocally) {
```
print "Local DB\n"
}

if (rd.canModify) {

    print "May be modified by current user\n"
}

p2 = rd.project

print fullName p "\n"

for rmd in rd do {

    print "\nModules present in definition :\n"

    mv = rmd.moduleVersion
    print fullName mv "\t"

    print rmd.dataConfigView "\t"
    print rmd.ddcView "\t"

    if (rmd.createdLocally) {

        print "Home DB.\n"
    }

    ddc = rmd.ddcMode

    if (ddc == ddcFullModule){

        print "Module will not be editable once definition is exported.\n"
    } else if (ddc == ddcByObject){


print "Selected objects will be locked in the local database once the
definition is exported.\n"

} else if (ddcm == ddcByAttribute){

    print "Selected attributes will be locked in the local database once
the definition is exported.\n"

} else if (ddcm == ddcReadOnly){

    print "Module will only be editable in the local database once
definition is exported.\n"

} else if (ddcm == ddcNone){

    print "Module will be fully editable in both local and target
databases when definition is exported.\n"

}

if (rd.name == "RifDef1"){

    s = exportPackage (rd, stm, myDB, b)

    if (s != ""){

        print "Error occurred : " s "\n"

    }

}
The following example dumps all information about all RIF imports in the current project. It then merges those imports where required.
RifImport ri
RifDefinition rd
Project p = current
User importer, merger
string importerName, mergerName, res
Folder f
Skip dates = create

for rd in p do {

    for ri in rd do {

        rd = ri.definition
        print rd.name "\n"

        f = ri.folder
        print "Located in : " fullName f
        print "\n"

        importer = ri.importedBy
        importerName = importer.name
        print "Imported by :  " importerName "\n"

        print "Imported on : " ri.importTime "\n"

        if (ri.mergeStarted && !ri.mergeCompleted) {
            print "Merge started on : " ri.mergeTime "\n"

        } else if (ri.mergeCompleted) {

            print "Merge completed on : " ri.mergeTime "\n"

        }
    }
}
if (ri.mergeRequired) {
    print "Merge required.\n"
    res = rifMerge (ri, null)
    print "Merging result : " res "\n"
} else {

    merger = ri.mergedBy
    print "Merged by : " mergerName "\n"
}

if (ri.mergeDisabled) {

    print "Merge disabled, locks removed.\n"
}
print "\n"
Chapter 8

New in DXL for Rational DOORS 9.1

This chapter describes features that are new in Rational DOORS 9.1:

- Regular Expressions

## Regular Expressions

**regexp2**

**Declaration**

```dxl
Regexp regexp2(string expression)
```

**Operation**

Creates a regular expression. Its behavior will not be changed to match the legacy behavior of `regexp()` . Should be used in all new regular expression code.
Chapter 9

New in DXL for Rational DOORS 9.0

This chapter describes features that are new in Rational DOORS 9.0:

- Discussions
- Descriptions
- Filtering
- HTML
- Miscellaneous

Discussions

- Discussion Types
- Properties
- Iterators
- Operations
- Triggers
- Example

Discussion Types

Discussion

Represents a discussion.

Comment

Represents a comment in a discussion.

DiscussionStatus

Represents the status of a discussion. The possible values are Open and Closed.
Properties

The following tables describe the properties available for the discussion and comment types. Property values can be accessed using the . (dot) operator, as shown in the following syntax:

`variable.property`

where:

- `variable` is a variable of type `Discussion` or `Comment`
- `property` is one of the discussion or comment properties

### Discussion

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>DiscussionStatus</td>
<td>The status of the discussion: whether it is open or closed.</td>
</tr>
<tr>
<td>summary</td>
<td>string</td>
<td>The summary text of the discussion, which may be null</td>
</tr>
<tr>
<td>createdBy</td>
<td>User</td>
<td>The user who created the discussion, if it was created in the current database. Otherwise it returns null.</td>
</tr>
<tr>
<td>createdByName</td>
<td>string</td>
<td>The name of the user who created the discussion, as it was when the discussion was created.</td>
</tr>
<tr>
<td>createdByFullName</td>
<td>string</td>
<td>The full name of the user who created the discussion, as it was when the discussion was created.</td>
</tr>
<tr>
<td>createdOn</td>
<td>Date</td>
<td>The date and time the discussion was created.</td>
</tr>
<tr>
<td>createdDataTimestamp</td>
<td>Date</td>
<td>The last modification timestamp of the object or module that the first comment in the discussion referred to.</td>
</tr>
<tr>
<td>lastModifiedBy</td>
<td>User</td>
<td>The user who added the last comment to the discussion, or who last changed the discussion status</td>
</tr>
<tr>
<td>lastModifiedByName</td>
<td>string</td>
<td>The user name of the user who added the last comment to the discussion, or who last changed the discussion status</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Extracts</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>lastModifiedByFullName</td>
<td>string</td>
<td>The full name of the user who added the last comment to the discussion, or who last changed the discussion status.</td>
</tr>
<tr>
<td>lastModifiedOn</td>
<td>Date</td>
<td>The date and time the last comment was added, or when the discussion status was last changed.</td>
</tr>
<tr>
<td>lastModifiedDataTimestamp</td>
<td>Date</td>
<td>The last modification timestamp of the object or module that the last comment in the discussion referred to.</td>
</tr>
<tr>
<td>firstVersion</td>
<td>ModuleVersion</td>
<td>The version of the module the first comment was raised against.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> If a comment is made against the current version of a module and the module is then baselined, this property will return a reference to that baseline. If the baseline is deleted, it will return the deleted baseline.</td>
</tr>
<tr>
<td>lastVersion</td>
<td>ModuleVersion</td>
<td>The version of the module the latest comment was raised against. See note for the <code>firstVersion</code> property above.</td>
</tr>
<tr>
<td>firstVersionIndex</td>
<td>string</td>
<td>The baseline index of the first module version commented on in the discussion. Can be used in comparisons between module versions.</td>
</tr>
<tr>
<td>lastVersionIndex</td>
<td>string</td>
<td>The baseline index of the last module version commented on in the discussion. Can be used in comparison between module versions.</td>
</tr>
</tbody>
</table>
Comment

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>string</td>
<td>The plain text of the comment.</td>
</tr>
<tr>
<td>moduleVersionIndex</td>
<td>string</td>
<td>The baseline index of the module version against which the comment was raised. Can be used in comparisons between module versions.</td>
</tr>
<tr>
<td>status</td>
<td>DiscussionStatus</td>
<td>The status of the discussion in which the comment was made.</td>
</tr>
<tr>
<td>moduleVersion</td>
<td>ModuleVersion</td>
<td>The version of the module against which the comment was raised.</td>
</tr>
<tr>
<td>onCurrentVersion</td>
<td>bool</td>
<td>True if the comment was raised against the current version of the module or an object in the current version.</td>
</tr>
<tr>
<td>changedStatus</td>
<td>bool</td>
<td>Tells whether the comment changed the status of the discussion when it was submitted. This will be true for comments that closed or re-opened a discussion.</td>
</tr>
<tr>
<td>dataTimestamp</td>
<td>Date</td>
<td>The last modified time of the object or module under discussion, as seen at the commenting users client at the time the comment was submitted.</td>
</tr>
<tr>
<td>createdBy</td>
<td>User</td>
<td>The user that created the comment. Returns null if the user is not in the current user list.</td>
</tr>
<tr>
<td>createdByName</td>
<td>string</td>
<td>The user name of the user who created the comment, as it was when the comment was created.</td>
</tr>
<tr>
<td>createdByFullName</td>
<td>string</td>
<td>The full name of the user who created the comment, as it was when the comment was created.</td>
</tr>
<tr>
<td>createdOn</td>
<td>Date</td>
<td>The data and time when the comment was created.</td>
</tr>
<tr>
<td>discussion</td>
<td>Discussion</td>
<td>The discussion containing the comment.</td>
</tr>
</tbody>
</table>

Note: If a comment if made against the current version of a module and the module is then baselined, this property will return a reference to that baseline. If the baseline is deleted, it will return the deleted baseline.

Iterators
for Discussion in Type

Syntax

for disc in Type do {
  ...
}

where:

- disc is a variable of type Discussion
- Type is a variable of type Object, Module, Project or Folder

Operation

Assigns the variable disc to be each successive discussion in Type in the order they were created. The first time it is run the discussion data will be loaded from the database.

The Module, Folder and Project variants will not include discussions on individual objects.

The Folder and Project variants are provided for forward compatibility with the possible future inclusion of discussions on folders and projects. They perform no function in Rational DOORS 9.0.

for Comment in Discussion

Syntax

for comm in disc do {
  ...
}

where:

- comm is a variable of type Comment
- disc is a variable of type Discussion

Operation

Assigns the variable comm to be each successive comment in disc in chronological order. The first time it is run on a discussion in memory, the comments will be loaded from the database. Note that if a discussion has been changed by a refresh (e.g. in terms of the last Comment timestamp) then this will also refresh the comments list.

The discussion properties will be updated in memory if necessary, to be consistent with the updated list of comments.
Operations

create(Discussion)

Declaration

string create(target, string text, string summary, Discussion& disc)

Operation

Creates a new Discussion about target, which can be of type Object or Module. Returns null on success, error string on failure. Also add text as the first comment to the discussion.

addComment

Declaration

string addComment(Discussion disc, target, string text, Comment& comm)

Operation

Adds a Comment about target to an open Discussion. Note that target must be an Object or Module that the Discussion already relates to. Returns null on success, error string on failure.

closeDiscussion

Declaration

string closeDiscussion(Discussion disc, target, string text, Comment& comm)

Operation

Closes an open Discussion disc by appending a closing comment, specified in text. Note that target must be an Object or Module that disc already relates to. Returns null on success, error string on failure.

reopenDiscussion

Declaration

string reopenDiscussion(Discussion disc, target, string text, Comment& comm)

Operation

Reopens a closed Discussion disc and appends a new comment, specified in text. Note that target must be an Object or Module that disc already relates to. Returns null on success, error string on failure.
deleteDiscussion

Declaration

string deleteDiscussion(Discussion d, Module m|Object o)

Operation

Deletes the specified module or object discussion if the user has the permission to do so. Returns null on success, or an error string on failure.

sortDiscussions

Declaration

void sortDiscussions({Module m|Object o|Project p|Folder f}, property, bool ascending)

Operation

Sorts the discussions list associated with the specified item according to the given property, which may be a date, or a string property as listed in the discussions properties list. String sorting is performed according to the lexical ordering for the current user’s default locale at the time of execution.

If the discussion list for the specified item has not been loaded from the database, this perm will cause it to be loaded.

The Folder and Project forms are provided for forward compatibility with the possible future inclusion of discussions on folders and projects. They perform no function in 9.0.

getDiscussions

Declaration

string getDiscussions({Module m|Object o|Project p|Folder f})

Operation

Refreshes from the database the Discussion data for the specified item in memory. Returns null on success, or an error on failure.

getObjectDiscussions

Declaration

string getObjectDiscussions(Module m)

Operation

Refreshes from the database all Discussions for all objects in the specified module. Returns null on success, or an error on failure.
getComments

Declaration

string getComments(Discussion d)

Operation

Refreshes from the database the comments data for the specified Discussion in memory. Returns null on success, or an error on failure.

Note: The Discussion properties will be updated if necessary, to be consistent with the updated comments list.

mayModifyDiscussionStatus

Declaration

bool mayModifyDiscussionStatus(Discussion d, Module m)

Operation

Checks whether the current user has rights to close or re-open the specified discussion on the specified module.

baselineIndex

Declaration

string baselineIndex(Module m)

Operation

Returns the baseline index of the specified Module, which may be a baseline or a current version. Can be used to tell whether a Comment can be raised against the given Module data in a given Discussion.

Note: A Comment cannot be raised against a baseline index which is less than the lastVersionIndex property of the Discussion.

Triggers

Trigger capabilities have been expanded so that triggers can now be made to fire before or after a Discussion or a Comment is created.

As follows:

<table>
<thead>
<tr>
<th></th>
<th>pre</th>
<th>post</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
**comment**

**Declaration**

Comment comment(Trigger t)

**Operation**

Returns the Comment with which the supplied Trigger is associated, null if not a Comment trigger.

**discussion**

**Declaration**

Discussion discussion(Trigger t)

**Operation**

Returns the Discussion with which the supplied Trigger is associated, null if not a Discussion trigger.

**dispose(Discussion/Comment)**

**Declaration**

void dispose({Discussion& d|Comment& c})

**Operation**

Disposes of the supplied Comment or Discussion reference freeing the memory it uses.

Can be called as soon as the reference is no longer required.

**Note:** The disposing will take place at the end of the current context.

**Example**

```dxl
// Create a Discussion on the current Module, with one follow-up Comment...
Module m = current
Discussion disc = null
create(m,"This is my\n first comment.","First summary",disc)
Comment cmt
```
addComment(disc, m, "This is the\nsecond comment.", cmt)

// Display all Discussions on the Module
for disc in m do
{
    print disc.summary " (" disc.status ")\n"
    User u = disc.createdBy
    string s = u.name
    print "Created By: " s "\n"
    print "Created By Name: \"" disc.createdByName "\"\n"
    print "Created On: " stringOf(disc.createdOn) "\n"
    u = disc.lastModifiedBy
    s = u.name
    print "Last Mod By: " s "\n"
    print "Last Mod By Name: \"" disc.lastModifiedByName "\"\n"
    print "Last Mod On " stringOf(disc.lastModifiedOn) "\n"
    print "First version: " (fullName disc.firstVersion) " [" //-
        (versionString disc.firstVersion) "]\n"
    print "Last version: " (fullName disc.lastVersion) " [" //-
        (versionString disc.lastVersion) "]\n"
    Comment c
    for c in disc do
    {
        print "Comment added by " (c.createdByName) " at " //-
            (stringOf(c.createdOn)) ":\n"
        print "Module Version: " (fullName c.moduleVersion) " [" //-
            (versionString c.moduleVersion) "]\n"
        print "Data timestamp: " (stringOf c.dataTimestamp) "\n"
        print "Status: " c.status " (" (c.changedStatus ? "Changed" //-
            : "Unchanged") ")\n"
        print "On current: " c.onCurrentVersion "\n"
        print c.text "\n"
    }
}

DXL Reference Manual
Descriptions

This section describes the DXL support in Rational DOORS for the new description functionality.

- View Descriptions
- Attribute Type Descriptions
- Attribute Definition Descriptions

View Descriptions

setViewDescription

Declaration

void setViewDescription(ViewDef vd, string desc)

Operation

Sets the description for a view where *vd* is the view definition handle.

getViewDescription

Declaration

string getViewDescription(ViewDef vd)

Operation

Returns the description for a view where *vd* is the view definition handle.

Attribute Type Descriptions

setDescription

Declaration

AttrType setDescription(AttrType at, string desc, string &errMess)

Operation

Sets the description for the specified attribute type. Returns null if the description is not successfully updated.
modify

Declaration

AttrType modify(AttrType at, string name, string codes[], int values, int colors, string desc[], [int arrMaps[]], string &errMess)

Operation

Modifies the supplied attribute type with the corresponding values and descriptions. Can be used to update the descriptions of old enumeration types.

The optional arrMaps argument specifies existing index values for enumeration values, taking into consideration their re-ordering.

create

Declaration

AttrType create(string name, string codes[], int values[], int colors[], string desc[], string &errMess)

Operation

The new desc[] argument enables the creation of a new enumeration based attribute type, whose enumerations use those descriptions. Returns null if creation is not successful.

description property

Both attribute types themselves, and the enumeration values they may contain, have a new description property. It can be accessed by using the dot (.) operator.

Example

AttrType at
string desc
int i
...
//To get the description of the attribute type
desc = at.description
...
//To get the description of the enumeration values with index i
desc = at.description[i]
Attribute Definition Descriptions

description property

Attribute definitions can now contain a description property. It can be accessed by using the dot (.) operator.

Example

Module m = current
AttrDef ad = find(m, "AttrName")
print ad.description

description(create)

Attribute definition descriptions can be specified during their creation.

Example

AttrDef ad = create object (description "My description") (type "string") //-
   (default "defvalue") (attribute "AttrName")

description(modify)

Attribute definition descriptions can be altered by using the modify perm is one of the following ways. Note the new setDescription property constant.

Example1

Module m = current
AttrDef ad = find(m, "AttrName")
modify (ad, module (description "New Description") (type "string") //-
   (default "New default") (attribute "New Name"))

Example2

Module m = current
AttrDef ad = find(m, "AttrName")
modify (ad, setDescription, "New description text")
Filtering

This section describes the DXL support in Rational DOORS for the new module explorer filtering functionality added in Rational DOORS 9.0.

applyFiltering

Declaration
void applyFiltering(Module)

Operation
Sets the module explorer display to reflect the current filter applied to the specified module.

unApplyFiltering

Declaration
void unApplyFiltering(Module)

Operation
Switches off filtering in the module explorer for the specified module.

applyingFiltering

Declaration
bool applyingFiltering(Module)

Operation
Returns a boolean indicating whether filtering is turned on in the module explorer for the specified module.

HTML

This section describes the DXL support the HTML functionality added in Rational DOORS 9.0.

- HTML Control
- HTML Edit Control
HTML Control

The section describes the DXL support for the HTML control added in Rational DOORS 9.0.

**Note:** Some of the functions listed below take an ID string parameter to identify either a frame or an HTML element. In each of these methods, frames or elements nested within other frames are identified by concatenating the frame IDs and element IDs as follows: `<top frame ID>/[<sub frame ID>/...]<element ID>`.

In methods requiring a frame ID, passing `null` into this parameter denotes the top level document.

These methods refer to all frame types including IFRAME and FRAME elements.

### htmlView

**Declaration**

```dxl
DBE htmlView(DB parentDB, int width, int height, string URL, bool before_navigate_cb(DBE element, string URL, string frame, string postData), void document_complete_cb(DBE element, string URL), bool navigate_error_cb(DBE element, string URL, string frame, int statusCode), void progress_cb(DBE element, int percentage))
```

**Operation**

Creates an HTML view control where the arguments are defined as follows:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>parentDB</strong></td>
<td>The dialog box containing the control.</td>
</tr>
<tr>
<td>width</td>
<td>The initial width of the control.</td>
</tr>
<tr>
<td>height</td>
<td>The initial height of the control.</td>
</tr>
<tr>
<td>URL</td>
<td>The address that will be initially loaded into the control. Can be null to load a blank page (about:blank).</td>
</tr>
</tbody>
</table>
**before_navigate_cb**  
Fires for each document/frame before the HTML window/frame navigates to a specified URL. It could be used, amongst other things, to intercept and process the URL prior to navigation, taking some action and possibly also navigating to a new URL. The return value determines whether to cancel the navigation. Returning `false` cancels the navigation.

Its arguments are defined as follows:
- **element**: The HTML control itself.
- **URL**: The address about to be navigated to.
- **frame**: The frame for which the navigation is about to take place.
- **postData**: The data about to be sent to the server if the HTTP POST transaction is being used.

**document_complete_cb**  
Fires for each document/frame once they are completely loaded and initialized. It could be used to start functionality required after all the data has been received and is about to be rendered, for example, parsing the HTML document.

Its arguments are defined as follows:
- **element**: The HTML control itself.
- **URL**: The loaded address.

**navigate_error_cb**  
Fires when an error occurs during navigation. Could be used, for example, to display a default document when internet connectivity is not available.

The return value determines whether to cancel the navigation. Returning `false` cancels the navigation.

Its arguments are defined as follows:
- **elements**: The HTML control itself.
- **URL**: The address for which navigation failed.
- **frame**: The frame for which the navigation failed.
- **statusCode**: Standard HTML error code.

**progress_cb**  
Used to notify about the navigation progress, which is supplied as a percentage.
set(html callback)

Declaration

```c
void set(DBE HTMLView, bool event_cb(DBE element, string ID, string tag, string event_type))
```

Operation

Attaches a callback to HTML control element that receives general HTML events. The ID argument identifies the element that sourced the event, the tag argument identifies the type of element that sourced the event, and the event_type argument identifies the event type. Note that the only event types currently supported are click and dblclick.

If this function is used with an incorrect DBE type, a DXL runtime error occurs.

set(html URL)

Declaration

```c
void set(DBE HTMLView, string URL)
```

Operation

Navigates the given HTMLView to the given URL.

Can only be used to navigate the top level document and cannot be used to navigate nested frame elements.

setURL

Declaration

```c
void setURL(DBE HTMLView, string ID, string URL)
```

Operation

Navigates the frame identified by ID to the given URL. The ID may be null.

getURL

Declaration

```c
string getURL(DBE HTMLView, string ID)
```

Operation

Returns the URL for the currently displayed frame as identified by its ID. The ID may be null.
get(HTML view)

Declaration

```
string get(DBE HTMLView)
```

Operation

Returns the URL currently displayed in the given `HTMLView`, if there is one.

get(HTML frame)

Declaration

```
Buffer get(DBE HTMLView, string ID)
```

Operation

Returns the URL for the currently displayed frame as identified by its `ID`.

set(HTML view)

Declaration

```
string set(DBE HTMLView, Buffer HTML)
```

Operation

Sets the HTML fragment to be rendered inside the `<body>` tags by the HTML view control directly. This enables the controls HTML to be constructed dynamically and directly rendered.

setHTML

Declaration

```
string setHTML(DBE HTMLView, string ID, Buffer HTML)
```

Operation

Sets the HTML fragment to be rendered inside the `<body>` tags by the HTML view controls frame as identified by `ID`. This enables the HTML of the given document or frame to be constructed dynamically and directly rendered.

Note: The contents of the frame being modified must be in the same domain as the parent HTML document to be modifiable. A DXL error will be given on failure (for example, if the wrong type of DBE is supplied).
### getHTML

**Declaration**

```dxl
Buffer getHTML(DBE HTMLView, string ID)
```

**Operation**

Returns the currently rendered HTML fragment inside the `<body>` tags of the document or frame as identified by its `ID`.

### getBuffer

**Declaration**

```dxl
Buffer getBuffer(DBE HTMLView)
```

**Operation**

Returns the currently rendered HTML.

### getInnerText

**Declaration**

```dxl
string getInnerText(DBE HTMLView, string ID)
```

**Operation**

Returns the text between the start and end tags of the first object with the specified `ID`.

### setInnerText

**Declaration**

```dxl
void setInnerText(DBE HTMLView, string ID, string text)
```

**Operation**

Sets the text between the start and end tags of the first object with the specified `ID`.

### getInnerHTML

**Declaration**

```dxl
string getInnerHTML(DBE HTMLView, string ID)
```

**Operation**

Returns the HTML between the start and end tags of the first object with the specified `ID`. 
setInnerHTML

**Declaration**

void setInnerHTML(DBE HTMLView, string ID, string html)

**Operation**

Sets the HTML between the start and end tags of the first object with the specified ID.

**Note:** The innerHTML property is read-only on the col, colGroup, framSet, html, head, style, table, tBody, tFoot, tHead, title, and tr objects.

getAttribute

**Declaration**

string getAttribute(DBE element, string ID, string attribute)

**Operation**

Retrieves the value for the requested attribute of the first object with the specified value of the ID attribute. If the attribute does not exist, null is returned.

Returns null on success. Returns error string on failure, for example if the wrong type of DBE is passed in.

setAttribute

**Declaration**

void setAttribute(DBE element, string ID, string attribute)

**Operation**

Sets the value of the requested attribute for the first object with the specified value of the ID attribute. If the attribute does not exist, it is added to the object.

Displays a DXL error on failure, for example if the wrong type of DBE is passed in.

**Example**

```dxl
DB dlg
DBE htmlCtrl
DBE htmlBtn
DBE html

void onTabSelect(DBE whichTab) {
```
int selection = get whichTab
}

void onSetHTML(DBE button) {
    Buffer b = create
    string s = get(htmlCtrl)
    print s
    b = s
    set(html, b)
    delete b
}

void onGetInnerText(DBE button) {
    string s = getInnerText(html, "Text")
    confirm(s)
}

void onGetInnerHTML(DBE button) {
    string s = getInnerHTML(html, "Text")
    confirm(s)
}

void onGetAttribute(DBE button) {
    string s = getAttribute(html, "Text", "Align")
    confirm(s)
}

void onSetInnerText(DBE button) {
    Buffer b = create
    string s = get(htmlCtrl)
    setInnerText(html, "Text", s)
}

void onSetInnerHTML(DBE button) {
Buffer b = create
string s = get(htmlCtrl)
setInnerHTML(html, "Text", s)
}

void onSetAttribute(DBE button){
    Buffer b = create
    string s = getAttribute(html, "Text", "Align")
    if (s == "left"){
        s = "center"
    }
    else if (s == "center"){
        s = "right"
    }
    else if (s == "right"){
        s = "left"
    }

    setAttribute(html, "Text", "align", s)
}

bool onHTMLBeforeNavigate(DBE dbe, string URL, string frame, string body){
    string buttons[] = {"OK"}
    string message = "Before navigate - URL: " URL "\nFrame: " frame "\nPostData: " body "\n"
    print message ""
    return true
}

void onHTMLDocComplete(DBE dbe, string URL){
    string buttons[] = {"OK"}
    string message = "Document complete - URL: " URL "\n"
    print message ""
    string s = get(dbe)
bool onHTMLError(DBE dbe, string URL, string frame, int error) {
    string buttons[] = {"OK"}
    string message = "Navigate error - URL: " URL "; Frame: " frame "; Error: " error "
    print message "
    return true
}

void onHTMLProgress(DBE dbe, int percentage) {
    string buttons[] = {"OK"}
    string message = "Percentage complete: " percentage "%"
    print message
    return true
}

dlg = create("Test", styleCentered | styleThemed | styleAutoparent)
htmlCtrl = text(dlg, "Field:", "<html><body><p id="Text">
align="center">Welcome to <b>DOORS <i>ERS</i></b></p>
</body></html>", 200, false)
htmlBtn = button(dlg, "Set HTML...", onSetHTML)
DBE getInnerTextBtn = button(dlg, "Get Inner Text...", onGetInnerText)
DBE getInnerHTMLBtn = button(dlg, "Get Inner HTML...", onGetInnerHTML)
DBE getAttributeBtn = button(dlg, "Get Attribute...", onGetAttribute)
DBE setInnerTextBtn = button(dlg, "Set Inner Text...", onSetInnerText)
DBE setInnerHTMLBtn = button(dlg, "Set Inner HTML...", onSetInnerHTML)
DBE setAttributeBtn = button(dlg, "Set Attribute...", onSetAttribute)

DBE frameCtrl = frame(dlg, "A Frame", 800, 500)

string strTabLabels[] = {"One","Two"}
DBE tab = tab(dlg, strTabLabels, 800, 500, onTabSelect)
htmlCtrl->"top"->"form"
htmlCtrl->"left"->"form"
htmlCtrl->"right"->"unattached"
htmlCtrl->"bottom"->"unattached"
htmlBtn->"top"->"spaced"->htmlCtrl
htmlBtn->"left"->"form"
htmlBtn->"right"->"unattached"
htmlBtn->"bottom"->"unattached"
getInnerTextBtn->"top"->"spaced"->htmlCtrl
getInnerTextBtn->"left"->"spaced"->htmlBtn
getInnerTextBtn->"right"->"unattached"
getInnerTextBtn->"bottom"->"unattached"
getInnerHTMLBtn->"top"->"spaced"->htmlCtrl
getInnerHTMLBtn->"left"->"spaced"->getInnerTextBtn
getInnerHTMLBtn->"right"->"unattached"
getInnerHTMLBtn->"bottom"->"unattached"
getAttributeBtn->"top"->"spaced"->htmlCtrl
getAttributeBtn->"left"->"spaced"->getInnerHTMLBtn
getAttributeBtn->"right"->"unattached"
getAttributeBtn->"bottom"->"unattached"
setInnerTextBtn->"top"->"spaced"->htmlBtn
setInnerTextBtn->"left"->"aligned"->getInnerTextBtn
setInnerTextBtn->"right"->"unattached"
setInnerTextBtn->"bottom"->"unattached"
setInnerHTMLBtn->"top"->"spaced"->htmlBtn
setInnerHTMLBtn->"left"->"spaced"->setInnerTextBtn
setInnerHTMLBtn->"right"->"unattached"
setInnerHTMLBtn->"bottom"->"unattached"
HTML Edit Control

The section describes the DXL support for the HTML edit control added in Rational DOORS 9.0.

The control behaves in many ways like a rich text area for entering formatted text. It encapsulates its own formatting toolbar enabling the user to apply styles and other formatting.
htmlEdit

Declaration

DBE htmlEdit(DB parentDB, string label, int width, int height)

Operation

Creates an HTML editor control inside parentDB.

htmlBuffer

Declaration

Buffer getBuffer(DBE editControl)

Operation

Returns the currently rendered HTML fragment shown in the control. The fragment includes everything inside the <body> element tag.

set(HTML edit)

Declaration

void set(DBE editControl, Buffer HTML)

Operation

Sets the HTML to be rendered by the edit control. The HTML fragment should include everything inside, but not including, the <body> element tag.

Example

DB MyDB = create "hello"
DBE MyHtml = htmlEdit(MyDB, "HTML Editor", 400, 100)

void mycb (DB dlg){

    Buffer b = getBuffer MyHtml
    string s = stringOf b
    ack s

}
apply (MyDB, "GetHTML", mycb)
set (MyHtml, "Initial Text")
show MyDB

Miscellaneous

delete(regexp)

Declaration
void delete(Regexp)

Operation
New in Rational DOORS 9.0 this perm deletes the supplied regular expression and frees the memory used by it.

getURL

Declaration
string getURL(Database__ d)
string getURL(Module m)
string getURL(ModName_ modName)
string getURL(ModuleVersion modVer)
string getURL(Object o)
string getURL(Folder f)
string getURL(Project p)
string getURL(Item i)

Operation
Returns the Rational DOORS URL of the given parameter.

backSlasher

Declaration
buffer backSlasher(Buffer b)
Operation

This function takes a buffer and converts all forward-slash characters (/) to back-slash characters (\), eliminates any repeated back-slash characters, and removes any trailing back-slash characters.

Example

```markdown
string s = "\\directory///file 
Buffer b = create
b = s
b = backSlasher(b)
print b 
```

Chapter 10
Fundamental types and functions

This chapter describes the functions and operators that can be used on the fundamental types of the core language underlying DXL:

- Operations on all types
- Operations on type bool
- Operations on type char
- Operations on type int
- Operations on type real
- Operations on type string

Operations on all types

The concatenation operator and the functions print and null can be used with all fundamental types.

Concatenation (base types)

The space character is the concatenation operator, which is shown as \(<\text{space}>\) in the following syntax:

```plaintext
bool b <space> string s
real r <space> string s
char c <space> string s
int i <space> string s
string s1 <space> string s2
```

<table>
<thead>
<tr>
<th>For type</th>
<th>A space character</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>Concatenates string (s) onto the evaluation of (b) (true or false), and returns the resulting string.</td>
</tr>
<tr>
<td>real</td>
<td>Concatenates string (s) onto real number (r), and returns the resulting string.</td>
</tr>
<tr>
<td>char</td>
<td>Concatenates the string (s) onto the character (c) and returns the result as a string.</td>
</tr>
<tr>
<td>int</td>
<td>Concatenates the string (s) onto the integer (c) and returns the result as a string.</td>
</tr>
<tr>
<td>string</td>
<td>Concatenates string (s_2) onto string (s_1) and returns the result as a string.</td>
</tr>
</tbody>
</table>
Concatenation must be used when printing derived types. An example of a derived type is \texttt{o."Object text"}, where \texttt{o} is an object. If a string is not concatenated to the end of the print statement, a DXL error will occur, in this case.

**Example**

```dxl
print "square root of 2 is " (sqrt 2.0) "\n"
char nl = '\n'
print "line one" nl "line two"
print (getenv "DOORSHOME") "/lib/dxl"
print o."Object text" ""
```

<table>
<thead>
<tr>
<th>Declaration</th>
<th>Print</th>
</tr>
</thead>
<tbody>
<tr>
<td>void print(bool \texttt{x})</td>
<td></td>
</tr>
<tr>
<td>void print(real \texttt{r})</td>
<td></td>
</tr>
<tr>
<td>void print(char \texttt{c})</td>
<td></td>
</tr>
<tr>
<td>void print(int \texttt{i})</td>
<td></td>
</tr>
<tr>
<td>void print(string \texttt{s})</td>
<td></td>
</tr>
</tbody>
</table>

**Operation**

<table>
<thead>
<tr>
<th>For type</th>
<th>Prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{bool}</td>
<td>The string \texttt{true} in the DXL output window if \texttt{x} is \texttt{true}; otherwise prints \texttt{false}.</td>
</tr>
<tr>
<td>\texttt{real}</td>
<td>The passed real number \texttt{r} in the DXL output window, using a precision of 6 digits after the radix character.</td>
</tr>
<tr>
<td>\texttt{char}</td>
<td>The character \texttt{c} in the DXL output window.</td>
</tr>
<tr>
<td>\texttt{int}</td>
<td>Integer \texttt{i} in the DXL output window, with a trailing newline.</td>
</tr>
<tr>
<td>\texttt{string}</td>
<td>The string \texttt{s} in the DXL output window without a trailing newline.</td>
</tr>
</tbody>
</table>

**Example**

```dxl
print (2.2 * 2.2) // prints 4.840000
print 'a'
print "Hello world\n"
```

<table>
<thead>
<tr>
<th>null</th>
</tr>
</thead>
</table>

The null function either returns the null value for the type, or tests whether a variable has the null value for its type.
Declaration

type null()
bool null(type x)

Operation

The first form returns the following values depending on the value of type:

<table>
<thead>
<tr>
<th>Type</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>char</td>
<td>character of ASCII code 0</td>
</tr>
<tr>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>real</td>
<td>0.000000</td>
</tr>
<tr>
<td>string</td>
<td>a null string (&quot;&quot;)</td>
</tr>
</tbody>
</table>

The second form returns true if x has a null value as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Null value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>false or null</td>
</tr>
<tr>
<td>char</td>
<td>null</td>
</tr>
<tr>
<td>int</td>
<td>0 or null</td>
</tr>
<tr>
<td>real</td>
<td>Any 0 value with any number of decimal places or null</td>
</tr>
<tr>
<td>string</td>
<td>&quot;&quot; or null</td>
</tr>
</tbody>
</table>

You can use the value null to assign a null value to any type, including type bool and char.

Example

string empty = null
print null empty   // prints true

Operations on type bool

Just as C++ has introduced a separate type bool (for boolean), so has DXL.

See also “Concatenation (base types),” on page 113, the print function, and the null function.
Type bool constants

The following constants are declared:
const bool true
const bool on
const bool false
const bool off

The boolean value true is equivalent to on; the value false is equivalent to off.

Note: For boolean values you cannot use 1 and 0.

Boolean operators

The operators &&, ||, and ! perform logical AND, OR, and NOT operations, as shown in the following syntax:
bool x && bool y
bool x || bool y
!bool x

These operators use lazy evaluation.
The && operator returns true only if x and y are both true; otherwise, it returns false. If x is false, it does not evaluate y.
The || operator returns true if x or y is true; otherwise, it returns false. If x is true, it does not evaluate y.
The ! operator returns the negation of x.

Type bool comparison

Type bool relational operators can be used as shown in the following syntax:
bool x == bool y
bool x != bool y

The == operator returns true only if x and y are equal; otherwise, it returns false.
The != operator returns true only if x and y are not equal; otherwise, it returns false.

Operations on type char

See also “Concatenation (base types),” on page 113, the print function, and the null function.
Character comparison

Character relational operators can be used as shown in the following syntax:

```dxl
char ch1 == char ch2
char ch1 != char ch2
char ch1 < char ch2
char ch1 > char ch2
char ch1 <= char ch2
char ch1 >= char ch2
```

These operators return true if ch1 is equal, not equal, less than, greater than, less than or equal to, or greater than or equal to ch2.

Character extraction from string

The index notation, [ ], can be used to extract a single character from a string, as shown in the following syntax:

```dxl
string text[int n]
```

This returns the n\textsuperscript{th} character of string text, counting from 0.

Example

This example prints h in the DXL Interaction window’s output pane:

```dxl
string s = "hello"
char c = s[0]
print c
```

Character classes

The set of functions whose names start with is can be used to check whether a character belongs to a specific class.

Declaration

```dxl
bool isalpha(char ch)
bool isupper(char ch)
bool islower(char ch)
bool isdigit(char ch)
bool isxdigit(char ch)
bool isalnum(char ch)
bool isspace(char ch)
```
bool ispunct(char ch)
bool isprint(char ch)
bool iscntrl(char ch)
bool isascii(char ch)
bool isgraph(char ch)

**Operation**

These functions return `true` if the character `ch` is in the named character class:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>'a' - 'z' 'A' - 'Z'</td>
</tr>
<tr>
<td>upper</td>
<td>'A' - 'Z'</td>
</tr>
<tr>
<td>lower</td>
<td>'a' - 'z'</td>
</tr>
<tr>
<td>digit</td>
<td>'0' - '9'</td>
</tr>
<tr>
<td>xdigit</td>
<td>'0' - '9' 'a' - 'f' 'A' - 'F'</td>
</tr>
<tr>
<td>alnum</td>
<td>'a' - 'z' 'A' - 'Z' '0' - '9'</td>
</tr>
<tr>
<td>space</td>
<td>' ' '	' '\n' '\m' '\j' '\k'</td>
</tr>
<tr>
<td>punct</td>
<td>any character except &lt;space&gt; and alpha numeric characters</td>
</tr>
<tr>
<td>print</td>
<td>a printing character</td>
</tr>
<tr>
<td>cntrl</td>
<td>any character code between 0 and 31, and code 127</td>
</tr>
<tr>
<td>ascii</td>
<td>any character code between 0 and 127</td>
</tr>
<tr>
<td>graph</td>
<td>any visible character</td>
</tr>
</tbody>
</table>

**Example**

print isalpha 'x' // prints true
print isalpha ' ' // prints false

**charOf**

**Declaration**

char charOf(int asciiCode)

**Operation**

Returns the character whose ASCII code is `asciiCode`.
Example

const char nl = charOf 10

intOf (char)

Declaration

int(char ch)

Operation

Returns the ASCII code of character ch.

Example

print intOf 'a' // prints 97

Operations on type int

A type int value in DXL has at least 32 bits.

See also “Concatenation (base types),” on page 113, the print function, and the null function.

Arithmetic operators (int)

Arithmetic operators can be used as shown in the following syntax:

int x + int y
int x - int y
int x * int y
int x / int y
int x % int y
int x | int y
int x & int y
~int x
-int x

These operators perform integer arithmetic operations for addition, subtraction, multiplication, division, remainder, bitwise OR, bitwise AND, bitwise NOT, and negation.
Assignment (int)

Assignment operators can be used as shown in the following syntax:

```plaintext
int x = int y
int x += int y
int x -= int y
int x *= int y
int x /= int y
int x %= int y
int x |= int y
int x &= int y
```

These operators assign integer values to variables of type `int`. Assignment. The last seven variations combine an arithmetic operation with the assignment.

**Example**

```plaintext
int y = 20
y *= 3
print y // print 60
y /= 7
print y // print 8
y %= 3
print y // print 2
```

**Unary operators**

Unary operators can be used to increment or decrement variables before or after their values are accessed, as shown in the following syntax:

```plaintext
int x++
int x--
int ++x
int --x
```

The first two operators return the value of the variable before incrementing or decrementing a variable. The second two return the value after incrementing or decrementing a variable.

**Note:** You can overload these operators.
Example

```dxl
int i = 40
print ++i       // prints 41
print i++       // prints 41
print i         // prints 42
```

Minimum and maximum operators

Two operators can be used to obtain the minimum or maximum value from a pair of integers, as shown in the following syntax:

```dxl
int x <? int y
int x >? int y
```

These operators return the minimum or maximum of integers \( x \) and \( y \).

Example

```dxl
print (3 <? 2)     // prints 2
print (3 >? 2)     // prints 3
```

Integer comparison

Integer relational operators can be used as shown in the following syntax:

```dxl
int x == int y
int x != int y
int x <  int y
int x >  int y
int x <= int y
int x >= int y
```

These operators return \( \text{true} \) if \( x \) is equal, not equal, less than, greater than, less than or equal to, or greater than or equal to \( y \).

Example

```dxl
print (2 != 3)  // prints true
```

isValidInt

**Declaration**

```dxl
bool isValidInt(string value)
```
Operation

Returns `true` if `value` is a valid integer; otherwise, returns `false`. The value passed must not be just spaces, e.g. " ".

If a null string is passed, a DXL run-time error occurs.

random(int)

Declaration

```dxl
int random(int max)
```

Operation

Returns a random integer value `x` such that `0 <= x < max`

Example

```dxl
print random 100 // prints an integer in the range 0 to 99
```

Operations on type real

A type `real` value in DXL is like a type `double` in C, with a precision of 64 bits.

See also “Concatenation (base types),” on page 113, the `print` function, and the `null` function.

Type real pi

The only constant of type `real` that is declared in DXL is `pi`:

```dxl
const real pi
```

This supplies a constant value of 3.141593.

Arithmetic operators (real)

Arithmetic operators can be used as shown in the following syntax:

```dxl
real x + real y
real x - real y
real x * real y
real x / real y
real x ^ real y
-real x
```
Operation

These operators perform arithmetic operations on type real variables for addition, subtraction, multiplication, division, exponentiation, and negation.

Example

```plaintext
print (2.2 + 3.3)      // prints 5.500000
```

Assignment (real)

Assignment operators can be used as shown in the following syntax:

```plaintext
real x = real y
real x += real y
real x -= real y
real x *= real y
real x /= real y
```

These operators perform type real assignment. The last four variations combine an arithmetic operation with the assignment.

Example

```plaintext
real x = 1.1
print (x += 2.0)        // prints 3.1
```

After the print statement, the variable x is assigned the value 3.1.

Convert to real

The assignment operator = can be used to convert an integer to a real number, as shown in the following syntax:

```plaintext
real r = int i
```

Operation

Converts i into a type real, assigns it to the type real variable r, and returns this value.

Example

```plaintext
real r = 5
print r          // prints 5.000000
```

Type real comparison

Type real relational operators can be used as shown in the following syntax:

```plaintext
real x == real y
```
real x != real y
real x <  real y
real x >  real y
real x <= real y
real x >= real y

These operators return true if x is equal, not equal, less than, greater than, less than or equal to, or greater than or equal to y.

Example
print (2.2 < 4.0)      // prints true

intOf (real)

Declaration
int intOf(real r)

Operation
Rounds r of type real to the nearest integer.

Example
print intOf 3.2      // prints 3

realOf

Declaration
real realOf(int i)
real realOf(string s)

Operation
Converts type int i or type string s into a type real value, and returns it.

Example
print realOf 4       // prints 4.000000
real x = realOf "3.2"
print x              // prints 3.200000
cos

Declaration
real cos(real angle)

Operation
Returns the cosine of angle in radians.

sin

Declaration
real sin(real angle)

Operation
Returns the sine of angle in radians.

tan

Declaration
real tan(real angle)

Operation
Returns the tangent of angle in radians.

exp

Declaration
real exp(real x)

Operation
Returns the natural exponent of type real x.

log

Declaration
real log(real x)

Operation
Returns the natural logarithm of type real x.
pow

Declaration
real pow(real x, 
        real y)

Operation
Returns type real x raised to the power y (same as x^y).

sqrt

Declaration
real sqrt(real x)

Operation
Returns the square root of x.

random(real)

Declaration
real random()

Operation
Returns a random value x, such that 0 <= x < 1.

Operations on type string

A DXL type string can contain any number of characters.
See also “Concatenation (base types),” on page 113, the print function, and the null function.

String comparison

String relational operators can be used as shown in the following syntax:

string s1 == string s2
string s1 != string s2
string s1 < string s2
string $s1 >$ string $s2$
string $s1 <=$ string $s2$
string $s1 >=$ string $s2$

These operators return true if $s1$ is equal, not equal, less than, greater than, less than or equal to, or greater than or equal to $s2$. Case is significant.

**Example**

```dxl
print ("aaaa" < "a" ) // prints "false"
print ("aaaa" > "a" ) // prints "true"
print ("aaaa" == "a" ) // prints "false"
print ("A" > "a" ) // prints "false"
print ("McDonald" < "Man" ) // prints "false"
```

---

**Substring extraction from string**

The index notation, [ ], can be used to extract a substring from a string, as shown in the following syntax:

```dxl
string text[range]
```

**Operation**

Returns a substring of text as specified by range, which must be in the form int:int.

The range argument is specified as the indices of the first and last characters of the desired substring, counting from 0. If the substring continues to the end of the string, the second index can be omitted.

**Example**

```dxl
string str = "I am a string constant"
print str[0:3] // prints "I am"
print str[2:3] // prints "am"
print str[5:] // prints "a string constant"
```

---

**cistrcmp**

**Declaration**

```dxl
int cistrcmp(string $s1,
             string $s2)
```

**Operation**

Compares strings $s1$ and $s2$ without regard to their case, and returns:

```
0 if $s1 == s2$
```
Example
print cistrcmp("aAa","AaA")   // prints 0
print cistrcmp("aAa","aA")    // prints 1
print cistrcmp("aAa","aAaa")  // prints -1

length

Declaration
int length(string str)

Operation
Returns the length of the string str.

Example
print length "123"   // prints 3

lower, upper

Declaration
string lower(string str)
string upper(string str)

Operation
Converts and returns the contents of str into lower or upper case.

Example
string mixed = "aaaBBBcccc"
print lower mixed   // prints "aaabbbccccc"
print upper mixed   // prints "AAABBBCCCC"

soundex

Declaration
string soundex(string str)
Operation
Returns the soundex code of the string \textit{str}. Initial non-alphabetic characters of \textit{str} are ignored.

Soundex codes are identical for similar-sounding English words.

Example
Both these examples print \texttt{R265} in the DXL Interaction window’s output pane.
\begin{verbatim}
print (soundex "requirements")
print (soundex "reekwirements")
\end{verbatim}

\textbf{backSlasher}

\textbf{Declaration}
\begin{verbatim}
buffer backSlasher(Buffer b)
\end{verbatim}

\textbf{Operation}
This function takes a buffer and converts all forward-slash characters (/) to back-slash characters (\), eliminates any repeated back-slash characters, and removes any trailing back-slash characters.

Example
\begin{verbatim}
string s = "\\directory\\\\file 
Buffer b = create
b = s
b = backSlasher(b)
print b 
\end{verbatim}

\textbf{findPlainText}

\textbf{Declaration}
\begin{verbatim}
bool findPlainText(string s, string sub, int &offset, int &length, bool matchCase[, bool reverse])
\end{verbatim}

\textbf{Operation}
Returns true if string \textit{s} contains the substring \textit{sub}.

Both \textit{s} and \textit{sub} are taken to be plain text string. Use \textit{findRichText} to deal with strings containing RTF markup.

If \textit{matchCase} is true, string \textit{s} must contain string \textit{sub} exactly with matching case; otherwise, any case matches.

The function returns additional information in \textit{offset} and \textit{length}. The value of \textit{offset} is the number of characters in \textit{s} to the start of the first match with string \textit{sub}. The value of \textit{length} contains the number of characters in the matching string.
If `reverse` is specified and is true, then the search is started at the end of the string, and the returned values of `offset` and `length` will reflect the last matching string in `s`.

**Example**

```c
string s = "This shall be a requirement"
string sub = "shall"

int offset = null
int length = null
bool matchCase = true
bool reverse = true

if (findPlainText (s, sub, offset, length, matchCase, reverse)){
    print offset " : " length "\nprints "5 : 5"
}
```

**unicodeString**

**Declaration**

```c
string unicodeString(RTF_string__ str, bool convertAllSymbols, bool returnAsPlainText)
```

**Operation**

Returns the value of the specified rich text string as RTF or plain text. If the attribute contains characters in Symbol font, these characters are converted to the Unicode equivalents.

If `convertAllSymbols` is true, all symbol character are converted. If false, only Unicode characters that have a good chance of being displayed are used. See the `symbolToUnicode` perm for a description of which characters are converted.

The value is returned as plain text if `returnAsPlainText` is true. Otherwise the value is returned as RTF.

**escape**

**Declaration**

```c
string escape(string str, char escapeChar, string escapeChars)
```

**Operation**

Escapes all the characters in `str` which are in `escapeChars`, with the `escapeChar` character. This also escapes `escapeChar` itself.
Example

escape("hello world", '/', "l") returns "he/l/lo wor/ld"
escape("hello world #1", '#', "lh") returns "#he#l#lo wor#ld #1"

stripPath

Declaration

string stripPath(string path, bool isEscaped)

Operation

Removes the path part from path, using forward slash as the path separator.
If isEscaped is true, the slash character can be used as a literal character rather than a path separator by preceding the character with a backslash.

Example

stripPath("abc/def/ghi", b) returns "ghi", where b is true or false.
stripPath("abc/def/\\/ghi", true) returns "def/ghi"
Chapter 11

General language facilities

This chapter introduces basic functions and structures defined by DXL’s run-time environment, as follows:

• Files and streams
• Configuration file access
• Dates
• Skip lists
• Regular expressions
• Text buffers
• Arrays

Files and streams

This section describes DXL’s features for manipulating files. For information on creating a directory, see the *mkdir* function.

The main data type introduced is the *Stream*, which uses C++ like overloading of >> and << to read and write files. Streams are not a fundamental type inherited from DXL’s C origins, so the type name *Stream* begins with an upper case letter.

Standard streams

Declaration

*Stream& cin*
*Stream& cout*
*Stream& cerr*

Operation

Following C++’s naming scheme for UNIX standard streams, these variables are initialized by Rational DOORS to standard input, output and error.

On UNIX platforms, you can use *cin* to read input that has been piped into Rational DOORS, and *cout* to pipe data out from Rational DOORS. Similarly, you can send user defined error messages (or any other desired output) to standard error using *cerr*. 

DXL Reference Manual
Read from stream

The operator >> can be used to read strings or data from a configuration area stream, or fill a buffer, as shown in the following syntax:

file >> string s
file >> char c
file >> real r
file >> int i

file >> Buffer b

where:

file is a file of type Stream

The first form reads a line of text from the configuration area stream file into string s, up to but not including any newline.

The next three forms read the data from the configuration area stream file, and return the result as a stream, to enable chained reads. Real and integer constants are expected to be the last items on a line, while characters, including newlines, are read one at a time up to and including the end of file.

The second form reads from the configuration area stream file into buffer b until it is full at its current size, or the end of the file is reached. Returns the configuration area stream. This function can read multiple lines.

Example

char   c
real   r
int    i
Stream input = read "data.dat"
input >> c >> r >> i

Read line from stream

Two operators can be used to read a single line from a stream to a buffer, as shown in the following syntax:

file -> Buffer b
file >= Buffer b

where:

file is a file of type Stream

Operation

The -> operator reads a single line from the stream file, and copies it to the buffer, skipping any leading white space. If the line is empty besides white space, the buffer is emptied. Returns the stream.
The >= operator reads a single line from the stream file, and copies it to the buffer in its entirety. If the line is empty, the buffer is emptied. Returns the stream.

**Write to stream**

The operator << can be used to write strings, single characters or buffers to a stream, as shown in the following syntax:

```dxl
file << string s
file << char c
file << Buffer b
```

where:

- `file` is a file of type Stream

Writes the string s, the character c, or the buffer b to the stream file. To write other data types to a stream, first convert them to a string by concatenating the empty string or a newline.

**Example**

```dxl
Stream out = write tempFileName
out << 1.4 "\n"
Stream alpha = write tempFileName
alpha << 'a' << 'b' << 'c'
```

**canOpenFile**

**Declaration**

```dxl
bool canOpenFile(string pathname,
                 bool forWrite)
```

**Operation**

Returns true when the file pathname can be opened; otherwise, returns false. If `forWrite` is set to true, the file is opened for write and the current contents of the file are cleared. If `forWrite` is set to false the file is opened read only and the existing contents are unchanged.

**read, write, append(open file)**

**Declaration**

```dxl
Stream read(string filename)
Stream write(string filename)
Stream append(string filename)
```
Operation

Opens a file *filename* for reading, writing or appending, and returns a stream. File I/O operations only succeed if the user has permission to create or access the files specified.

To open a binary file, you must call the *binary* function after the *read*, *write* or *append*. The syntax is therefore:

```
read [binary] filename
write [binary] filename
append [binary] filename
```

You can use the Stat DXL functions to check whether the I/O functions in this section can succeed (see “user, size, mode,” on page 183).

Example

```
// ASCII file
Stream output = write tempFileName

// binary file
Stream image = read binary pictureFileName
```

close(stream)

Declaration

```
void close(Stream s)
```

Operation

Closes the stream *s*.

flush

Declaration

```
void flush(Stream s)
```

Operation

Flushes the output stream *s*. Character I/O can be buffered; this command forces any such buffers to be cleared.

readFile

Declaration

```
string readFile(string filename)
```

Operation

Returns the contents of the file *filename* as a string.
Note: The Codepages function also has a readFile operator. For information about Codepages and readFile, see “readFile,” on page 207.

---

goodFileName

**Declaration**

```dxl
goodFileName(string filename)
```

**Operation**

Returns a legitimate file name of the passed file, `filename`, with respect to any restrictions imposed by the current platform. This will only apply to the filename up to the `'.'` character. The string after the `'.'` is ignored.

This function does not support non-English Unicode characters.

**Example**

This example prints the file name `Test_results` in the DXL output window:

```
print goodFileName "Test results"
```

---

tempFileName

**Declaration**

```dxl
tempFileName()
```

**Operation**

Returns a string, which is a legal file name on the current platform, and is not the name of an existing file. On UNIX platforms, returns a file name like `/tmp/DOORSaaoueF`; on Windows platforms, returns a file name like `C:\TEMP\DP2`. This file can be used for temporary storage by DXL programs.

---

currentDirectory

**Declaration**

```dxl
currentDirectory()
```

**Operation**

Returns the path name of the current working directory.

---

copyFile

**Declaration**

```dxl
copyFile(string sourceFileName, string destFileName)
```
Operation

Copies file sourceFileName to destFileName. If the operation succeeds, returns null; otherwise, returns an error message.

Example
copyFile("file1", "file2")

deleteFile

Declaration

string deleteFile(string filename)

Operation

Deletes the file named filename. If the operation succeeds, returns null; otherwise, returns an error message.

renameFile

Declaration

string renameFile(string old, string new)

Operation

Renames the file called old to new. If the operation succeeds, returns null; if it fails, returns an error message.

data

Declaration

bool end(Stream s)

Operation

Returns true if the stream has no more characters pending. The test should be made after a read, but before the read data is used:

Example

while (true) {
    input >> str  // read a line at a time; var set up
    if (end input) break // test after read but before
    print str "\n"  // variable str is used
}
format

Declaration

void format(Stream s, string text, int width)

Operation

Outputs string text to Stream s, formatting each word of the text with a ragged right margin in a column of width characters. If a word is too long for the specified column, it is continued on the next line.

Example

Stream out = write tempFileName
format(out, "DXL Reference Manual", 5)
close out

This generates the following in the temporary file:

DXL
Reference
Manual

for file in directory

Syntax

for s in directory "pathname" do {
   ... 
}

where:

   pathname     is the path of the directory
   s            is a string variable

Operation

Sets the string s to be each successive file name found in the directory pathname.

Example

This example prints a list of the files in directory C:\:

string x = "c:\"
string file
for file in directory x do {
    print file "\n"
}

Files and streams example program

This example creates a temporary file, writes some data to it, saves it, renames it, reads from the new file, and then deletes it:

// file (Stream) DXL example
/*
  example file I/O program
*/
string filename = tempFileName // get a scratch file
print "Writing to " filename "\n"
Stream out = write filename
out << 'x' "" // write a char (via a string)
out << 1.001 "\n" // a real (must be last thing on line)
out << 42 "\n" // an int (must be last thing on line)
out << "hello world\na second line\n" // a string
close out // write a file to read back in again
string oldName = filename
filename = tempFileName // get a new file name
renameFile(oldName, filename) // move the file we wrote earlier
print "Reading from " filename "\n"
Stream input = read filename
char c // declare some variable
real r
int i
input >> c
input >> r
input >> i
print c " " r " " i "\n" // check data type read/writes
string str // do rest line by line
while (true) {
    input >> str // read a line at a time
Configuration file access

This section describes the DXL features for manipulating configuration files. The data types used are ConfType and ConfStream. Many of these functions have a parameter ConfType area. The arguments that can be passed as ConfType area are as follows:

- confUser
- confSysUser
- confSystem
- confTemp

The confUser argument means the file is situated in an area specific to the current Rational DOORS user, or to the current system user if a project is not open.

The confSysUser argument means the file is situated in the configuration area for system users. This argument remains constant regardless of whether the user is logged into the project. For example, the Rational DOORS Tip Wizard uses a confSysUser file to store whether a user has opted to show Tips on startup.

The confSystem argument means the file is situated in a shared area accessible by all users.

The confTemp argument is similar to confSystem, but is generally used for storing temporary files.

If the function does not supply an area argument, confUser is used.

Read from stream

The operator >> can be used to read strings or data from a configuration area stream, or fill a buffer, as shown in the following syntax:

file >> string s
file >> Buffer b

where:

file is a file of type ConfStream

The first form reads a line of text from the configuration area stream file into string s, up to but not including any newline.

The second form reads from the configuration area stream file into buffer b until it is full at its current size, or the end of the file is reached. Returns the configuration area stream. This function can read multiple lines.
Read line from stream

Two operators can be used to read a single line from a configuration stream to a buffer, as shown in the following syntax:

\[
file \rightarrow Buffer \ b \\
file \geq Buffer \ b
\]

where:

- *file* is a file of type *ConfStream*

**Operation**

The \( \rightarrow \) operator reads a single line from the configuration area stream *file*, and copies it to the buffer, skipping any leading white space. If the line is empty besides white space, the buffer is emptied. Returns the stream.

The \( \geq \) operator reads a single line from the configuration area stream *file*, and copies it to the buffer in its entirety. If the line is empty, the buffer is emptied. Returns the stream.

Write to stream

The operator \( \ll \) can be used to write strings, single characters or buffers to a stream, as shown in the following syntax:

\[
file \ll string \ s \\
file \ll char \ c \\
file \ll Buffer \ b
\]

where:

- *file* is a file of type *ConfStream*

Writes the string *s*, the character *c*, or the buffer *b* to the configuration area stream *file*. To write other data types to a configuration area stream, first convert them to a string by concatenating the empty string or a newline.

**Example**

ConfStream out = write tempFileName
out \ll 1.4 \"\n\"

ConfStream alpha = write tempFileName
alpha \ll 'a' \ll 'b' \ll 'c'

confMkdir

**Declaration**

void confMkdir(string *dirName
               [,ConfType *area])
Operation

Creates the directory, \textit{dirName}, in either the default or the specified configuration area, \textit{area}.

---

**confDeleteDirectory**

**Declaration**

\[
\text{string confDeleteDirectory(string } \text{pathname, ConfType } \text{conf})
\]

**Operation**

Deletes the named directory in the specified \textit{ConfType} area (\textit{confSystem} or \textit{confUser}). On success it returns null; on failure it returns an error string.

---

**confRead**

**Declaration**

\[
\text{ConfStream confRead(string } \text{fileName}\]
\[
\text{[,ConfType } \text{area}]\]

**Operation**

Opens the specified file for reading, and returns the file handle. The file can be in either the default or the specified configuration area.

Detects the encoding of conf files by checking for the presence of a UTF-8 Byte Order Marker (BOM) at the start of the file. If it finds one, it assumes that the file is encoded in UTF-8. Otherwise, it assumes that the file is encoded according to the legacy codepage for the database. In either case, any values subsequently read from the file using the \texttt{ConfStream} \texttt{>>} operator or others are converted to Unicode, so the encoding of the file should not affect the functionality of any DXL scripts that use this perm.

---

**confWrite**

**Declaration**

\[
\text{ConfStream confWrite(string } \text{fileName}\]
\[
\text{[,ConfType } \text{area}]\]

**Operation**

Opens the specified file for writing, and returns the file handle. The file can be in either the default or the specified configuration area.

Any conf files created by this perm are encoded in UTF-8, enabling them to contain any Unicode strings.
confAppend

Declaration

ConfStream confAppend(string fileName
                   [,ConfType area])

Operation

Opens the specified file for appending, and returns the file handle. The file can be in either the default or the specified configuration area.

This perm converts any non-UTF-8 files to UTF-8 encoding before opening them for append. This enables any Unicode strings to be written to the file using the ConfStream << write operators.

confRenameFile

Declaration

string confRenameFile(string old,
                    string new
                   [,ConfType area])

Operation

Renames the file old to new in either the default or the specified configuration area.

Returns an error message string if the operation fails.

confCopyFile

Declaration

string confCopyFile(string source,
            string dest,
            ConfType area)

Operation

Copies source to dest in the specified configuration area. If the operation fails, it returns an error message.

confDeleteFile

Declaration

string confDeleteFile(string fileName
                   [,ConfType area])
Operation
Deletes the specified file in either the default or the specified configuration area. If the operation fails, it returns an error message.

confFileExists

Declaration
bool confFileExists(string fileName [,ConfType area])

Operation
Returns true if the specified file exists in either the default or the specified configuration area; otherwise, returns false.

close(configuration area stream)

Declaration
void close(ConfStream s)

Operation
Closes the configuration area stream s.

end(configuration area stream)

Declaration
bool end(ConfStream s)

Operation
Returns true if the stream has no more characters pending. The test should be made after a read, but before the read data is used:

Example
while (true) {
    input >> str     // read a line at a time; var set up
    if (end input) break // test after read but before
    print str "\n"       // variable str is used
}

DXL Reference Manual
for file in configuration area

Syntax

```
for s in confDirectory("dirname", [area]) do {
    ...
}
```

where:

- `dirname` is the name of the directory in `area`, or if `area` is omitted, in `confUser`
- `area` is a constant of type `ConfType`: `confUser`, `confSysUser`, `confSystem`, `confTemp`, or `confProjUser`
- `s` is a string variable

Operation

Sets the string `s` to be each successive file name found in the directory `pathname`.

Example

This example prints a list of the files in directory `test` in `confUser`:

```
string file
for file in confDirectory("test") do {
    print file "\n"
}
```

confUploadFile(source, dest [, conftype])

Declaration

```
string confUploadFile(string source, string dest [, conftype])
```

Operation

Uploads a file from the location on the client machine specified by `source`, to the file in the system conf area on the database server, specified by `dest`. It returns null on success. If the `dest` string contains double-periods ".." or specifies an invalid directory, then the perm reports an error and returns null. Otherwise, if the upload fails, the perm returns an error message.

The optional 3rd argument specifies the config area where the file should be sent. This defaults to the current user's config area (confUser). Files to be accessible to all users should be uploaded to the system config area, by specifying this argument as "confSystem".

Example

```
string message = confUploadFile("C:\temp\myprog.exe","myprog",confSystem)
```
if (!null message)
{
    warningBox(message)
}

confDownloadFile(source, dest [, conftype])

Declaration
string confDownloadFile(string source, string dest [, conftype])

Operation
Downloads a file from the location in the conf area on the database server, specified by dest, to the location on the client machine specified by source. It returns null on success. If the source string contains double-periods “..” then the perm reports an error and returns null. Otherwise, if the download fails, the perm returns an error message.

The optional 3rd argument specifies the config area from which the file should be copied. This defaults to the current user’s config area (confUser).

Example
string message = confDownloadFile("myprog","C:\temp\myprog2.exe",confSystem)
if (!null message)
{
    warningBox(message)
}

Dates

This section describes DXL’s features for manipulating dates.

Dates are not a fundamental type inherited from DXL’s C origins, so the type name Date begins with an upper case letter.

DXL Date data limits are from 1 Jan 1970, to 31 Dec 2102.

Note: The date values always refer to your system’s current time zone.

Concatenation (dates)

The space character is the concatenation operator, which is shown as <space> in the following syntax:

Date d <space> string s
Concatenates string \( s \) onto date \( d \) and returns the result as a string. It uses the long format date, or, if any operations dealing in seconds have occurred, the short format date with time added.

**Example**

This example prints `<01 January 1999>`:

```dxl
Date d = "1 Jan 99"
print "<"d">"
```

---

**Assignment (date)**

The assignment operator `=` can be used as shown in the following syntax:

```dxl
Date d = string datestr
```

Converts the string `datestr` into a date, assigns it to \( d \), and returns it as a result. Issues an error message if `datestr` is not in a valid date format. Ordinal numbers, for example 4th, are not recognized. Apart from that limitation, all date formats are valid, for example:

- `yyyy`, `dd mmm`
- `dd/mm/yy`
- `mm/dd/yy`

Time can be appended to a dates using the format `hh:mm:ss.ss`, provided the date is in the format `dd/mm/yy` or `mm/dd/yy`.

**Example**

This example prints `04 October 1961`:

```dxl
Date d1 = "4 Oct 1961"
print d1
```

---

**Date comparison**

Date relational operators can be used as shown in the following syntax:

```dxl
Date d1 == Date d2
Date d1 != Date d2
Date d1 < Date d2
Date d1 > Date d2
Date d1 <= Date d2
Date d1 >= Date d2
```

These operators return `true` if \( d1 \) is equal, not equal, less than, greater than, less than or equal to, greater than or equal to \( d2 \).
Example
This example prints false in the DXL Interaction window’s output pane:

Date d1 = "4 Oct 1961"
Date d2 = "10 Nov 1972"
print (d1 > d2)

print(date)

Declaration
void print(Date d)

Operation
Prints the date d in the DXL output window in long format, or, if any operations dealing in seconds have occurred, the short format date with time added.

Example
This example prints 04 October 1961:

Date d1 = "4 Oct 1961"
print d1

today

Declaration
Date today()

Operation
Returns today’s date. The value includes the exact time, but it is not printed using:
print today
The function call:
intOf today
returns the integer number of seconds since 1 Jan 1970, 00:00:00 GMT.

Example
This example prints the current date and time:
print dateOf intOf today

Note: Concatenating strings to the end of this statement may give unexpected results.
session

**Declaration**

`Date session()`

**Operation**

Returns the date on which the current Rational DOORS session began. The value includes the exact time in the same way as the `today` function.

**Example**

This example prints the date the current Rational DOORS session started:

```dxl
print session
```

intOf(date)

**Declaration**

`int intOf(Date d)`

**Operation**

Returns an integer corresponding to the number of seconds that have elapsed between the given date and 1 Jan 1970, 00:00:00 GMT.

When a `Date` data type is converted for dates on or after 1 Jan 2037, or before 1 Jan 1970, this function returns a result of -1.

**Example**

```dxl
print intOf today
```

dateOf

**Declaration**

`Date dateOf(int secs)`

**Operation**

Returns the date and time that is calculated as `secs` seconds since 1 Jan 1970, 00:00:00 GMT.

**Example**

```dxl
int minute = 60
int hour = 60 * minute
int day = 24 * hour
int year = 365 * day
```
int leapYear = 366 * day
print dateOf ((year * 2) + leapYear)

This generates the following in the DXL Interaction window’s output pane:
01/01/73 00:00:00

This is three years after 1 Jan 1970, 00:00:00 GMT, taking into account that 1972 was a leap year.

---

### stringOf

#### Declaration

```dxl
definition stringOf(Date d[, Locale l][, string s] )
```

#### Operation

This returns the string representation of the date value using the specified locale and format. If no locale is specified, the current user locale is used. If no format string or a null format string is specified, then if the date value includes time (hours:minutes:seconds), the default short date format for the locale will be used. Otherwise, a long date format will be used. The default short date format will be either that specified by the user using `setDateFormat(Locale)`, or, if no default short date format has been set by the user for the locale, the system default format.

---

### date

#### Declaration

```dxl
definition Date date(string s[, Locale l][,string s])
```

#### Operation

This returns the date value represented by the supplied string, interpreted according to the specified locale and format. The default locale is the current user locale. If no format string is supplied, the input string is parsed using first the user’s default short date format (if one has been specified for the locale), and then all the supported formats for the locale.

---

### for string in shortDateFormats

#### Declaration

```dxl
definition for string in shortDateFormats([Locale l])
```

#### Operation

This iterator returns the short date formats supported for the specified locale. If no locale is specified, it returns the short date formats supported for the current user locale.

The first format returned is the default short date format for the locale.
for string in longDateFormats

Declaration

for string in longDateFormats([Locale l])

Operation

This iterator returns the long date formats supported for the specified locale. If no locale is specified, it returns the long date formats supported for the current user locale.

The first format returned is the default long date format for the locale.

includesTime

Declaration

bool includesTime(Date d)

Operation

This returns true if the specified date value includes time information as well as date.

dateOnly

Declaration

Date dateOnly(Date d)

Operation

Returns a copy of the supplied date value, without any included time-of-day information (it returns a date-only value).

dateAndTime

Declaration

Date dateAndTime(Date d)

Operation

Returns a copy of the supplied date value including time-of-day data.

Example

print today()
prints 6 June 2010
print dateAndTime(today)
Example
The following example uses the new locale specific date format perms.

```dxl
// dates.dxl - dates and formats example
//*********************************************************************
void testFormat(Date dateValue, Locale loc, string format)
// DESCRIPTION: Checks that the stringOf and dateOf perms are true
//           inverses for the specified format.
{
    print "    format " format ": " stringOf(dateValue, loc, format) "\n"
}
//****************************************************************************
void testDate(Date dateValue, Locale loc)
// Tests stringOf and dateOf using default formats, and all supported formats.
{
    // Test default format
    string stringForm = stringOf(dateValue, loc)
    print "Default format: " stringForm "\n"

    // Test all supported formats
    string format
    print "Short formats:\n"
    for format in shortDateFormats(loc) do
    {
        testFormat(dateValue, loc, format)
    }
    print "Long formats:\n"
    for format in longDateFormats(loc) do
    {
        testFormat(dateValue, loc, format)
    }

    // Test abbreviations.
    print "Abbreviated names: " stringOf(dateValue, loc, "ddd, d MMM yy") "\n"
```
// Test all full names.
print "Full names: " stringOf(dateValue, loc, "dddd, d MMMM yyyy") "\n"

Locale loc = userLocale
print "\nLOCALE: " (name loc) "\n"
print "\nDATE ONLY:\n"
testDate(today,loc)
print "\nDATE AND TIME:\n"
testDate(dateAndTime(today),loc)

---

Skip lists

This section describes DXL’s features for manipulating skip lists.

Skip lists are an efficient dictionary like data structure. Since DXL does not support a C like struct feature, many DXL programs use skip lists as the building blocks for creating complex data structures.

Because DXL provides no garbage collection, it is important to delete skip lists that are no longer required, thereby freeing allocated memory.

Skip lists are not a fundamental type inherited from DXL’s C origins, so the type name Skip begins with an upper case letter.

create, createString(skip list)

Declaration
Skip create()
Skip createString()

Operation
Creates a new empty skip list and returns it.

It is very important, and it is the programmer’s responsibility to ensure that data and keys are consistently used when storing and retrieving from a skip list. For example, you can cause program failure by inserting some data into a skip list as an integer, then retrieving the data into a string variable and attempting to print it.

The keys used with the skip list can be of any type. However, comparison of keys is based on the address of the key, not its contents. This is fine for elements that are always represented by a unique pointer, for example, objects, modules, or skip lists, but care is needed with strings. This is because a string may not have a unique address, depending on whether it is literal or a computed string stored in a variable.
There are two ways of avoiding this problem. The first is to use the `createString` form of the function for a skip list with a string key. The alternative is to ensure that all literal strings used as keys are concatenated with the empty string.

**Example**

```dxl
Skip strKeys = create
put(strKeys, "literal" "", 1000)
```

### delete(skip list)

**Declaration**

```dxl
void delete(Skip s)
```

**Operation**

Deletes all of skip list `s`. Variables that have been given as keys or data are not affected. This operation does not set the skip list to null. If the user checks the list for null, this will produce a DXL run-time error. The user should set the skip list to null after deletion.

### delete(entry)

**Declaration**

```dxl
bool delete(Skip s, type key)
```

**Operation**

Deletes an entry in skip list `s` according to the passed `key`, which can be of any type. Variables that have been given as keys or data are not affected. Returns `false` if the key does not exist.

**Example**

```dxl
if (delete(numberCache, 1)) // delete absno 1
    ack "delete succeeded"
```

### find(entry)

**Declaration**

```dxl
bool find(Skip s, type1 key
           [,type2 &data])
```

**Operation**

Returns `true` if the passed `key`, of `type1`, has an entry in skip list `s`. The optional third argument sets the entry found to be `data` of `type2`. Both `type1` and `type2` can be any type.
Example

```dxl
if (find(numberCache, 1, o)) {
    string h = o."Object Heading"
    ack h
}
```

---

**key**

The key function is used only within the skip list for loop, as shown in the following syntax:

```dxl
(type key(Skip s))
```

**Operation**

Returns the key corresponding to the current element. The return value can be of any type, so a cast must precede the use of key.

**Example**

```dxl
Object o
for o in numberCache do {
    // must cast the key command.
    int i = (int key numberCache)
    print i
}
```

---

**put**

**Declaration**

```dxl
bool put(Skip s,
        type1 key,
        type2 data)
```

**Operation**

Returns true if the passed key and data are successfully inserted into the skip list s. Duplicate entries are not allowed, so the function returns false if an entry with the same key already exists. For this reason, an entry at an existing key must first be deleted before its data can be changed.

**Example**

```dxl
Skip s = create
put(s, 1, 20)
print put(s, 1, 30)
// prints 'false'
delete(s, 1)
print put(s, 1, 30)
```
// prints 'true', s(1) is now 30

for data element in skip list

Syntax

for dataElement in skiplist do {
   ...
}

where:

dataElement is a variable of any type
skiplist is a variable of type Skip

Operation

Sets entry to be each successive type data element of list.

Example

Object o

for o in numberCache do {
   string h = o."Object Heading"
   print h "\n"
}

Skip lists example program

In this example a skip list is used to store a mapping from absolute numbers to the corresponding Rational DOORS object:

// skip list example
/*
   simple skip list example: make a mapping
   from absolute numbers to objects, allowing
   fast lookup
*/
Skip numberCache = create // builds the skip list

Object o

int      n = 0               // count objects

for o in current Module do {  
   // cycle through all objects
   int absno = o."Absolute Number"
   // get the number
```
put(numberCache, absno, o)
// number is key, object is data
n++
} // for
// we now have a quick way of going from absolute numbers to objects:
if (n > 0) {
    int i
    for i in 1:20 do {
        int absno = 1 + random n
        // choose an absno at random
        if (find(numberCache, absno, o)) {
            // can we find it?
            string heading = o."Object Heading"
            print "## absno " has
            heading \"" heading "\"\n"
        } // if
    } // for
} // if
```

## Regular expressions

This section describes DXL’s features for using regular expressions.

Regular expressions are a mechanism for detecting patterns in text. They have many applications, including searching and simple parsing.

Regular expressions are not a fundamental type inherited from DXL’s C origins, so the type name `Regexp` begins with an upper case letter.

The following symbols can be used in Regexp expressions:

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Example</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>* zero or more occurrences</td>
<td>a*</td>
<td>any number of a characters, or none</td>
</tr>
<tr>
<td>+ one or more occurrences</td>
<td>x+</td>
<td>one or more x characters</td>
</tr>
<tr>
<td>. any single character except newline</td>
<td>.*</td>
<td>any number of any characters (any string)</td>
</tr>
<tr>
<td>\ escape (literal text char)</td>
<td>.</td>
<td>literally a . (dot) character</td>
</tr>
</tbody>
</table>
Application of regular expressions

The space character is an operator that applies a regular expression to a string or buffer; it is shown as `<space>` in the following syntax:

- `Regexp reg <space> string text`
- `Regexp reg <space> Buffer b`

**Operation**

Returns `true` if there is a match.

**Example**

```dxl
Regexp line = regexp2 ".*,"
while (line txt1) {
    ...
}
```
**match**

The `match` function returns a range for a match of a regular expression within a string or buffer, as shown in the following syntax:

\[
\text{Regexp } r = \text{regexp } "x(options1)y(options2)\ldots" \\
\{\text{string|Buffer} \text{ str } = "string" \}
\]

\[\text{str[match } n]\]

where:

- \( r, \text{str} \) are variables
- \( x, y \) are literal characters in a regular expression
- \( \text{options1}, \text{options2} \) are regular expression matching options
- \( \text{string} \) is a string or buffer
- \( n \) is an integer

**Operation**

When \( n=0 \), returns the range of \( \text{string} \). When \( n=1 \), returns the range of the match for \( \text{options1} \); when \( n=2 \), returns the match for \( \text{options2} \), and so on. The value for \( n \) is restricted to the range 0-9.

**Example**

This example detects and decomposes URLs:

\[
\text{Regexp URL } = \text{regexp2 } "(HTTP|http|ftp|FTP|file|FILE)://([^ \s\),;>"]\)*"
\]

\[\text{string txt3 } = "The ABC URL is http://www.abc.com; it may be..."
\]

\[
\text{if (URL txt3) } \{
\text{print txt3[match 0] }"\n" \quad // \text{whole match} \\
\text{print txt3[match 1] }"\n" \quad // \text{first section in } ()
\text{print txt3[match 2] }"\n" \quad // \text{second section in } ()
\}
\]

**matches**

**Declaration**

\[
\text{bool matches(string reg,} \\
\quad \text{string text)}
\]
Operation

Returns true if text matches reg. For repeated use, declaring and building a regular expression is more efficient.

Example

string txt = "xxxxyesuuuu"
if (matches("(yes|no)", txt)) print txt[match 0]

regexp

Declaration

Regexp regexp(string reg)

Operation

Returns a new regular expression, specified by string reg. For legacy support only, should not be used in new code. Replaced by regexp2().

Example

// matches any line except newline
Regexp line = regexp2 ".*"

start, end(of match)

Declaration

int start(int n)
int end(int n)

Operation

Return the position of the first and last characters of the nth match from a call to match. The value for n is restricted to the range 0-9.

Example

int firstNameLen = end 1

delete(regexp)

Declaration

void delete(Regexp)

Operation

This perm deletes the supplied regular expression and frees the memory used by it.
regexp2

Declaration

Regexp regexp2(string expression)

Operation

Creates a regular expression. Its behavior will not be changed to match the legacy behavior of regexp(). Should be used in all new regular expression code.

Regular expressions example program

// regular expression DXL example
/*
   examples of regular expression DXL
*/
Regexp line = regexp2 ".*"
// matches any character except newline
string txt1 = "line 1\nline 2\nline 3\n" // 3 line string
while (!null txt1 && line txt1) {
    print txt1[match 0] "\n" // match 0 is whole of match
    txt1 = txt1[end 0 + 2: ] // move past newline
}
// The following regular expression detects and decomposes URLs
Regexp URL = regexp2 "(HTTP|http|ftp|FTP|file|FILE)://([\^ \:\\,;>\"]*)"
string txt3 = "The ABC URL is http://www.abcinc.com, and may be..."
if (URL txt3) {
    print txt3[match 0] "\n" // whole match
    print txt3[match 1] "\n" // first bracketed section
    print start 1 // position 15 in txt3 (from 0)
    print end 1 // 18
    print start 2 // 22
    print end 2 // 34
}

DXL Reference Manual
Text buffers

The following functions enable the manipulation of DXL buffers. Buffers are a speed and memory efficient way of manipulating text within DXL applications. Their use is particularly encouraged in parsers and importers.

You should explicitly delete buffers with `delete` as soon as they are no longer needed in a script.

Buffers are not a fundamental type inherited from DXL’s C origins, so the type name `Buffer` begins with an upper case letter.

Because DXL provides no garbage collection, it is important to delete buffers that are no longer required, thereby freeing allocated memory.

Assignment (buffer)

The assignment operator `=` can be used as shown in the following syntax:

```
Buffer b = string s
```

or

```
Buffer b = h.oldValue
```

**Operation**

The first form sets the contents of buffer `b` to that of the string `s`. You can use a range in the assignment.

The second form sets the contents of the buffer to the history property `oldValue`. The buffer should be deleted after use.

**Note:** If you want to assign a buffer to a buffer, you must use the form `Buffer b = stringOf(a)`, otherwise, the address of `a` is given to `b` instead of its value.

Append operator

The append operator `+=` can be used as shown in the following syntax:

```
Buffer b += string s
Buffer b += char c
Buffer b += Buffer b
```

**Operation**

Appends the string, character, or buffer to the buffer `b`.

**Example**

This example prints `one1twox` in the DXL Interaction window’s output pane:

```
Buffer buf1 = create
Buffer buf2 = create
```
buf1 = "one"
buf2 = "two"
buf1 += "1"
buf1 += buf2
buf1 += 'x'

---

**Concatenation (buffers)**

The space character is the concatenation operator, which is shown as `<space>` in the following syntax:

Buffer b <space> string s

Concatenates string s onto the contents of buffer b and returns the result as a string. You can use a range in the concatenation.

**Example**

Buffer b = create
b = "aaa"
print b "zzz" // prints "aaazzz"

---

**Buffer comparison**

String relational operators can be used as shown in the following syntax:

Buffer b1 == Buffer b2
Buffer b1 != Buffer b2
Buffer b1 < Buffer b2
Buffer b1 > Buffer b2
Buffer b1 <= Buffer b2
Buffer b1 >= Buffer b2

These operators return true if b1 is equal, not equal, less than, greater than, less than or equal to, or greater than or equal to b2. Case is significant.

**Example**

Buffer b1 = create
Buffer b2 = create
b1 = "aaa"
b2 = "aza"
print (b1==b2) " " (b1!=b2) " " (b1<b2) " " (b1>b2) " "
print (b1<=b2) " " (b1>=b2) " " (b1>b2) "\n"
Read and write operators

The `>>` operator can be used to read a stream into a buffer and return the stream (see “Read from stream,” on page 134).
The `<<` operator can be used to write a buffer to a stream and return the stream (see “Read line from stream,” on page 134).
The `->` and `>=` operators can be used to read a single line from a file to a buffer, (see “Write to stream,” on page 135).

Character extraction from buffer

The index notation, `[]`, can be used to extract a single character from a buffer, as shown in the following syntax:

```dxl
Buffer b[int n]
```

This returns the \(n^{th}\) character of buffer \(b\), counting from 0.

Example

This example prints \(a\) in the DXL Interaction window’s output pane:

```dxl
Buffer b = "abc"
char c = b[0]
print c
```

Substring extraction from buffer

The index notation, `[]`, can be used to extract a substring from a buffer, as shown in the following syntax:

```dxl
Buffer b[range]
```

Operation

Returns a range of \(b\) as specified by \(range\), which must be in the form `int:int`.

The \(range\) argument is specified as the indices of the first and last characters of the desired range, counting from 0. If the range continues to the end of the buffer, the second index can be omitted. This function returns a buffer or string depending on the type assigned.

Example

```dxl
Buffer buf = create
buf = "abcdefg"
string s = buf[2:3]
print s     // prints cd
Buffer b = buf[4:5]
print b     // prints ef
```
combine

Declaration

```c
void combine(Buffer b1,
    Buffer b2,
    int start
    [,int finish])
```

Operation

Concatenates a substring of `b2` onto the contents of `b1`. The substring is from `start` to `finish`, or if `finish` is omitted, from `start` to the end of the buffer. This function provides a performance advantage over the assignment to buffer using the range option.

Example

```c
Buffer b1 = create, b2 = create
b1 = "zzz"
b2 = "abcdef"
combine(b1, b2, 3, 4)
print stringOf b1      // prints "zzzde"
```

contains

Declaration

```c
int contains(Buffer b,
    char ch
    [,int offset])
int contains(Buffer b,
    string word,
    int offset)
```

Operation

The first form returns the index at which the character `ch` appears in buffer `b`, starting from 0. If present, the value of `offset` controls where the search starts. For example, if `offset` is 1, the search starts from 2. If `offset` is not present, the search starts from 0. If `ch` does not appear after `offset`, the function returns -1.

The second form returns the index at which string `word` appears in the buffer, starting from 0, provided the string is preceded by a non-alphanumeric character. The value of the mandatory `offset` argument controls where the search starts. If `word` does not appear after `offset`, the function returns -1.
getDOSstring

Declaration

Buffer getDOSstring(Buffer b)

Operation

Returns a copy of the supplied Buffer, with a carriage-return character inserted before any newline character that is not already preceded by a carriage return.

create(buffer)

Declaration

Buffer create([int initSize])

Operation

Creates a buffer. A buffer has no intrinsic limit on its size; when a buffer becomes full it extends itself, if memory permits. The argument initSize specifies the initial size of the buffer. If no initial size argument is passed, this function creates a buffer that uses a default initial size of 255.

delete(buffer)

Declaration

void delete(Buffer &b)

Operation

Deletes the buffer b, and sets the variable b to null.

firstNonSpace

Declaration

int firstNonSpace(Buffer b)

Operation

Returns the index of the first non-space character in buffer b, or -1 if there is none.
keyword(buffer)

Declaration

```c
int keyword(Buffer b,
             string word,
             int offset)
```

Operation

Returns the index at which string `word` appears in buffer `b`, starting from character `offset`, provided that the string is neither preceded nor followed by a non-alphanumeric character. If `word` does not appear, the function returns -1.

This function is used to accelerate parsing of programming languages.

length(buffer get)

Declaration

```c
int length(Buffer b)
```

Operation

Returns the length of the buffer.

length(buffer set)

Declaration

```c
void length(Buffer b,
            int len)
```

Operation

Sets the length of a buffer. This is normally used for truncating buffers, but can also be used to lengthen them.

The DXL program is responsible for the content of the buffer.

Example

```c
Buffer buf = create
buf = "abcd"
length(buf, 2)
print "<" (stringOf buf) ">"  // prints "ab"
```
set(char in buffer)

**Declaration**

```c
void set(Buffer b,
         int n,
         char ch)
```

**Operation**

Sets the character at position \( n \) of buffer \( b \) to character \( ch \).

**Example**

```c
if (name[n] == '.') set(name, n, ';')
```

setempty

**Declaration**

```c
void setempty(Buffer b)
```

**Operation**

Empties buffer \( b \), but does not reclaim any space.

setupper, setlower

**Declaration**

```c
void setupper(Buffer b)
void setlower(Buffer b)
```

**Operation**

These functions convert the case of buffer \( b \) to upper or lower case.

stringOf(buffer)

**Declaration**

```c
string stringOf(Buffer b)
```

**Operation**

Returns the contents of buffer \( b \) as a string.

**Example**

```c
Buffer b = create
```
Buffers and regular expressions

Regular expressions can be applied to buffers in the same way as strings (see “Application of regular expressions,” on page 159). The regular expression functions start, end(of match), and match can also be used with buffers.

Example

Buffer buf = create
buf = "aaaabbcccc"
Regexp re = regexp2 "a*"
re buf               // apply regular expression
print buf[match 0]   // prints "aaaa"

search

Declaration

bool search(Regexp re,
          Buffer b,
          int start
          [,int finish])

Operation

Searches part of b using re. The search starts at start and continues until finish, or if finish is omitted, from start to the end of the buffer.

This function provides a performance advantage over the concatenation of regular expression to buffer with the range option.

Note that the match, end and start regular expression functions can be used to return offsets relative to start, not the start of the buffer.

It is possible when using this perm along with a complex regular expression, and a very large Buffer, that valid code will produce a run-time error detailing an “incorrect regular expression”.

Text buffers example program

// buffer DXL example
/*
 * example use of DXL buffers - place a border
 around a multi-line piece of text, e.g.:
 +-----------------+
 | the quick brown |
Buffer process(Buffer source) {
    Regexp line = regexp2 ".*"  // matches up to newline
    int from = 0
    int max = 0
    Buffer boxed = create, horiz = create
    while (search(line, source, from)) {
        // takes a line at a time from source
        int offset = end 0  // end of the match within source
        string match = source[from:from+offset]
        from += offset + 2  // move 'from' over any newline
        if (null match)       // we are done
            break
        max = max >? length match  // remember max line length
    }
    if (max==0) {  // no strings matched
        boxed = "++\n++"
    } else {
        horiz = "++"  // build a horizontal line
        int i
        for i in 1:max+2 do  // allow two spaces
            horiz += '-'
        horiz += '+'
        horiz += '+'
        from = 0  // reset offset
        boxed += horiz
        while (search(line, source, from)) {
            // rescan buffer
            int offset = end 0
            string match =
                source[from:from+offset]
            if (null match)
                break
            from += offset + 2
    }
Boxed += '|'  // add the vertical bars
boxed += ' '  
boxed += match
for i in 1 : max - length match + 1 do
    boxed += ' '  
    // add space to side of box
boxed += '|'  
boxed += '\n'
}
boxed += horiz
return boxed
}
Buffer text = create
Text = "The quick brown"  // build a test string
Text += ' \n'
Text += "fox jumped over"
Text += ' \n'
Text += "the lazy dog"
cout = write "buffer.tmp"
cout << process text  // print result

---

### Arrays

This section describes a dynamically sized two-dimensional array data type. An example of its use is in the Rational DOORS ASCII output generator in the tools library. As with skip lists, you must retrieve data into variables of the same data type as they were put into the array, or program failure may occur.

Because DXL provides no garbage collection, it is important to delete DXL's dynamic arrays that are no longer required, thereby freeing allocated memory.

Dynamic arrays are not a fundamental type inherited from DXL's C origins, so the type name **Array** begins with an upper case letter.

---

**create(array)**

**Declaration**

Array create(int x,
        int y)
Operation

Creates a dynamically sized array of initial bounds \((x, y)\). Following C conventions, the minimum co-ordinate is \((0,0)\), and the maximum co-ordinate is \((x-1, y-1)\). If an assignment is made to an array element outside these initial bounds, the array is automatically resized. When viewing arrays with the `printCharArray` function, the X axis grows left to right across the page, while the Y axis grows down the page.

Both arguments to `create` must be greater than or equal to 1.

Example

This example creates an array with 50 elements in the X direction accessed from \((0,0)\) to \((49,0)\), and only one element in the Y direction:

```dxl
Array firstArray = create(50, 1)
```

delete(array)

Decleration

```dxl
void delete(Array a)
```

Operation

Deletes array \(a\); stored contents are not affected.

get(data from array)

Decleration

```dxl
type get(Array a,
        int x,
        int y)
```

Operation

Returns the data, of any type, stored in array \(a\) at position \((x, y)\). You must retrieve the data into a variable of the same type as used when the data was put into the array. To ensure that this works unambiguously in the way intended, you should use a cast prefix to the `get` command.

Arrays are not just for fundamental types like strings and integers. You can store any DXL type in them, for example, objects, modules, skip lists, and even other arrays.

Example

This example uses a cast prefix to `get`:

```dxl
Array a = create(10, 10)
string str
int i
put(a, "a string", 3, 4)
put(a, 1000, 3, 5)
```
str = (string get(a,3,4))  // cast get as string
print str "\n"           // prints "a string"
i = (int get(a, 3, 5))    // cast get as int
print i                   // prints "1000"

This example stores an array in an array:
Array a = create(4,1)
Object obj = first current Module
Module mod = current
Skip skp = create
Array arr = create(1,1)
put(a, obj, 0, 0)
put(a, mod, 1, 0)
put(a, skp, 2, 0)
put(a, arr, 3, 0)
put(arr,"I was nested in a!", 0, 0)
Object objRef = (Object get(a,0,0))
Module modRef = (Module get(a,1,0))
Skip   skpRef = (Skip   get(a,2,0))
Array  arrRef = (Array get(a,3,0))
string str    = (string get(arrRef, 0, 0))
print str       // prints "I was nested in a!"

get(string from array)

Declaration

string get(Array a,
            int x,
            int y,
            int len)

Operation
Retrieves len characters as a string from a starting at position (x,y). This is the matching get command for putString.

Example
Array a = create(10,10)
putString(a, "a string", 2, 2)
string some = get(a, 4, 2, 3)
print some "\n"       // prints "str"

put(data in array)

**Declaration**

```c
void put(Array a,
         type data,
         int x,
         int y)
```

**Operation**

Puts `data`, of any type, into array `a` at position `(x,y)`. If the new position is outside `a`’s current bounds, `a` is resized to accommodate the new element.

**putString**

**Declaration**

```c
void putString(Array a,
               string s,
               int x,
               int y)
```

**Operation**

Puts the string `s` into the array `a` in such a way that its character contents are placed in X-direction adjacent elements starting at `(x,y)`. The original, or any other desired string can be rebuilt by using the argument string form of `get(a, x, y, len)`. The 3-argument form of `get` can be used to retrieve individual characters. Attempting to retrieve a character as a string causes program failure.

**printCharArray**

**Declaration**

```c
void printCharArray(Array a,
                    Stream s,
                    int x1,
                    int y1,
                    int x2,
                    int y2)
```

**Operation**

Sends the section of array `a` defined by the passed co-ordinates `x1,y1` and `x2,y2`, to the stream `s`.

**Example**

```c
Array a = create(20,5)
```
int x,y
for y in 0 : 4 do         // populate an array with a
    for x in 0 : 19 do    // block of # characters.
        put(a, '#', x, y)
Stream out = write "array.tmp"    // open a stream
printCharArray(a, out, 0, 0, 19, 4)  // write original block
out << "\n"
putString(a,"abc", 3, 1)        // insert a string
printCharArray(a, out, 0, 0, 19, 4)
// view change
out << "\n"
close out
Chapter 12

Operating system interface

This chapter describes three major packages of functions that allow Rational DOORS to communicate with the host operating system:

- Operating system commands
- Windows registry
- Interprocess communications
- System clipboard functions

Operating system commands

This section defines functions that interact with the operating system under which Rational DOORS is being run. For a DXL program to be portable between platforms, care is needed when using these facilities. The functions that use the `Stat` data type work on the `stat` API provided by the operating system, which enables DXL programs to determine the status of files and directories.

platform

Declaration

```dxl
string platform()
```

Operation

Returns the name of the current Rational DOORS platform, currently one of:

- **Linux®**
- **Solaris**
- **WIN32**

This function can be used to make programs portable between platforms.

Example

```dxl
string fileGoodName_(string root, extpc, extunix) {
    if (platform == "WIN32")
        return currentDirectory "\\
            goodFileName root extpc
```
else
    return (getenv "HOME") "/"
goodFileName root extunix
}

The function fileGoodName_, defined in $DOORSHOME/lib/dxl/utils/fileops.dxl uses platform to construct an appropriate file name for the current operating system. Using such functions enables DXL programs to be useful on all platforms. Literal file names in programs may not be portable. The path /tmp/dxl/myfile may work on a WIN32 platform, but c:\temp\dxl\myfile cannot work on a UNIX platform.

getMemoryUsage

Declaration

int getMemoryUsage()

Operation

Returns the Rational DOORS client memory usage in MB.

g.getenv

Declaration

string getenv(string var)

Operation

Returns the current value of the environment variable var, as set in the operating system. Both Windows and UNIX platforms support this mechanism.

Note: You should know about your operating system’s environment variables before using this function. If necessary, consult the operating system documentation.

Example

prompt call getenv("HOME")
prompt call getenv("DATA")
prompt call getenv("DOORSHOME")
prompt call getenv("DOORS_DATA")

The first two examples return the corresponding variable values in the registry.

The second two examples return the corresponding variable values used in a command-line shortcut to start Rational DOORS, if set. Otherwise, returns the values set in the registry.
hostname

Declaration

string hostname()

Operation

Returns a string, which is the name of the current host system.

fullHostname

Declaration

string fullHostname(void)

Operation

Gets the fully qualified hostname of the machine on which the perm is executed.

mkdir

Declaration

void mkdir(string dirName
            [,string osParm])

Operation

Creates directory dirName.

Optional argument osParm can contain information that is dependent on the operating system, such as the UNIX octal file access mask.

Example

The following example creates a typical UNIX path name, and sets the access rights:
mkdir("/usr/development/phase1", "0755")

The following example creates a Windows path, for which there are no access rights:
mkdir("C:\DOORS\DXLExample\", "")

setenv

Declaration

void setenv(string var,
            string s)
Operation

Sets the registry variable var to s in the registry section
HKEY_CURRENT_USER\Software\Telelogic\DOORS\<DOORS version>\Config, where <DOORS version> is the version number of the current version of Rational DOORS installed.

Before using this function, you should be familiar with your operating system’s registry variables. If necessary, consult your operating system documentation.

setServerMonitor

Declaration

void setServerMonitor(bool on)

Operation

On Windows platforms only, when on is true, activates the Rational DOORS Server Monitor. This inserts an icon in the Windows task bar that monitors client server communications.

serverMonitorIsOn

Declaration

bool serverMonitorIsOn()

Operation

On Windows platforms only, returns true if the Rational DOORS Server Monitor is active. Otherwise, returns false.

username

Declaration

string username()

Operation

Returns a string that contains the operating system defined user name under which Rational DOORS is being run. This may not be the same as the Rational DOORS user name returned by doorsname, depending on the current project’s setup.

system

Declaration

void system(string command)
Operation
On Windows platforms only, passes the string command to the operating system for execution, and continues the current DXL program. Using platform in conjunction with this function prevents an error message on UNIX platforms.

Example
if (platform=="WIN32")
    system "notepad"

Note that if the command to be executed is a built in DOS command, such as del, you need, for example:

system "c:\windows\command.exe /c del temp.txt"

Declaration
void system(string command,
    void childCB(int)
    [,void parentCB()])

Operation
On UNIX platforms only, passes the string command to the operating system for execution.

Unlike the Windows system function, these functions terminate the current execution path of the calling DXL program. One or two callback functions must be provided. In the first form, only a function childCB is needed. This function is called when the operating system finishes execution of command. In the second form, parentCB is also provided; this is called concurrently with the operating system’s processing of command, enabling the calling DXL program to continue work while the command is being executed.

Example
void cb(){
    print "system command executing\n"
}
void nullCB(int status){
}
if (platform == "WIN 32"){
    system("E:\winnt\system32\command.exe")
        cb
} else{
    system ("xterm", nullCB, cb)
}

create(status handle)

Declaration
Stat create(Stream s)
Stat create(string filename)
Operation

Returns a status handle for the stream or file name, which is used in the other `Stat` functions.

---

**delete(status handle)**

**Declaration**

```c
void delete(Stat s)
```

**Operation**

Deletes the handle \( s \).

---

**accessed, modified, changed(date)**

**Declaration**

```c
Date accessed(Stat s)
Date modified(Stat s)
Date changed(Stat s)
```

**Operation**

Returns the accessed, modified or changed date of the stream or file identified by the handle.

---

**directory, symbolic, regular**

**Declaration**

```c
bool directory(Stat s)
bool symbolic(Stat s)
bool regular(Stat s)
```

**Operation**

Returns `true` if the stream or file identified by the handle is a directory, a symbolic link, or a regular file respectively.

**Example**

```c
c
Stat s
string filename = "/etc"
s = create filename
if (!null s && directory s)
    ack filename " is a directory!"
```
user, size, mode

Declaration

```dxl
string user(Stat s)
int size(Stat s)
int mode(Stat s)
```

Operation

Returns the user name (PC file on windows), size, or mode of the stream or file identified by the handle.

The following constant integers are used with the `int mode(Stat)` function as bit-field values (using standard UNIX stat semantics).

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_ISUID</td>
<td>set user id on execution</td>
</tr>
<tr>
<td>S_ISGID</td>
<td>set group id on execution</td>
</tr>
<tr>
<td>S_IRWXU</td>
<td>read, write, execute permission: owner</td>
</tr>
<tr>
<td>S_IRUSR</td>
<td>read permission: owner</td>
</tr>
<tr>
<td>S_IWUSR</td>
<td>write permission: owner</td>
</tr>
<tr>
<td>S_IXUSR</td>
<td>execute/search permission: owner</td>
</tr>
<tr>
<td>S_IRWXG</td>
<td>read, write, execute permission: group</td>
</tr>
<tr>
<td>S_IRGRP</td>
<td>read permission: group</td>
</tr>
<tr>
<td>S_IWGRP</td>
<td>write permission: group</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>execute/search permission: group</td>
</tr>
<tr>
<td>S_IRWXO</td>
<td>read, write, execute permission: other</td>
</tr>
<tr>
<td>S_IROTH</td>
<td>read permission: other</td>
</tr>
<tr>
<td>S_IWOTH</td>
<td>write permission: other</td>
</tr>
<tr>
<td>S_IXOTH</td>
<td>execute/search</td>
</tr>
</tbody>
</table>

Example

The following example shows how to emulate the formatting of part of the UNIX command `ls -l`.

```dxl
string filename = "/etc"
Stat s = create filename
```
if (!null s) {
    int modes = mode s
    print (modes&S_ISUID!=0 ? 's' : '-')
    print (modes&S_IRUSR!=0 ? 'r' : '-')
    print (modes&S_IWUSR!=0 ? 'w' : '-')
    print (modes&S_IXUSR!=0 ? 'x' : '-')
    print (modes&S_IRGRP!=0 ? 'r' : '-')
    print (modes&S_IWGRP!=0 ? 'w' : '-')
    print (modes&S_IXGRP!=0 ? 'x' : '-')
    print (modes&S_IROTH!=0 ? 'r' : '-')
    print (modes&S_IWOTH!=0 ? 'w' : '-')
    print (modes&S_IXOTH!=0 ? 'x' : '-')
    print "\t" filename
"}

Status handle functions example

This example is taken from $DOORSHOME/lib/dxl/utils/fileops.dxl.
bool fileExists_(string filename) {
    Stat s
    s = create filename
    if (null s) return false
    delete s
    return true
}

It is used by several of the DXL Library tools to determine whether a file exists.

Windows registry

getRegistry

Declaration
    string getRegistry(string keyName,
                          string valueName)
Operation

Returns a string representation of the named value of the specified Windows registry key.

The keyName argument must be a fully specified registry key, beginning with any one of the following:

HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_CLASSES_ROOT
HKEY_USERS

If valueName is null, returns the default value for the key. If the key does not exist, the value does not exist, or the operating system is not a Windows platform, returns null.

Example

```java
string s = "HKEY_CURRENT_USER\SOFTWARE\Microsoft Office\9.3\Common\LocalTemplates"
print getRegistry(s, null) 
```

```java
string s = "HKEY_CURRENT_USER\SOFTWARE\Microsoft Office\95\WORD\OPTIONS"
print getRegistry(s, "DOC-PATH") 
```

setRegistry

Declaration

```java
string setRegistry(string keyName,
                   string valueName,
                   {string|int} value)
```

Operation

Sets the named value of the specified registry key to have the value supplied and the appropriate registry type, as follows:

<table>
<thead>
<tr>
<th>Type of value</th>
<th>Registry type</th>
</tr>
</thead>
<tbody>
<tr>
<td>string value</td>
<td>REG_SZ</td>
</tr>
<tr>
<td>integer value</td>
<td>REG_DWORD</td>
</tr>
</tbody>
</table>

The key is created if one does not already exist. If valueName is null, the default key value is set.

The keyName argument must be a fully specified registry key, beginning with any one of the following:

HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_CLASSES_ROOT
HKEY_USERS
This function is only usable on Windows platforms.
If the operation fails, returns an error message; otherwise returns null.

Example

```dxl
string s = "HKEY_CURRENT_USER\SOFTWARE\XYZ Inc.\The Product\Verification"
// Set default value of key
string errMess = setRegistry(s, null, "Default string value")
// Set named string value
errMess = setRegistry(s, "Configuration Parameter", "Is enabled")
// Set named integer value
checkStringReturn setRegistry(s, "Usage count", 1234)
```

deleteKeyRegistry

**Declaration**

```dxl
string deleteKeyRegistry(string keyName)
```

**Operation**

Deletes the named key from the registry, therefore extreme caution should be used.
The `keyName` argument must be a fully specified registry key, beginning with any one of the following:

HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_CLASSES_ROOT
HKEY_USERS

This function is only usable on Windows platforms.
If the operation fails, returns an error message; otherwise returns null.

**Example**

```dxl
// Clear up keys created
string errMess = deleteKeyRegistry "HKEY_CURRENT_USER\-
SOFTWARE\XYZ Inc.\The Product\Verification"
errMess = deleteKeyRegistry "HKEY_CURRENT_USER\SOFTWARE\XYZ Inc.\The Product"
errMess = deleteKeyRegistry "HKEY_CURRENT_USER\SOFTWARE\XYZ Inc."
```
deleteValueRegistry

Declaration

string deleteValueRegistry(string keyName, 
                          string valueName)

Operation

Deletes the named value from the specified registry key. If valueName is null, deletes the default value for the key.

Note:    Use caution when calling this function.

The keyName argument must be a fully specified registry key, beginning with any one of the following:

- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE
- HKEY_CLASSES_ROOT
- HKEY_USERS

This function is only usable on Windows platforms.

If the operation fails, returns an error message; otherwise returns null.

Example

string s = "HKEY_CURRENT_USER\SOFTWARE\XYZ Inc.-
            The Product\Verification"

// Delete named value
string errMess = deleteValueRegistry(s, "Usage count")

// Delete default value
errMess = deleteValueRegistry(s, null)

Interprocess communications

There are two forms of interprocess communications (IPC):

- The first uses TCP/IP. It can be used with the UNIX and Windows operating systems on all supported platforms.
- The second uses sockets, where a file is used to pass messages. It works only on UNIX platforms.

For examples of how to use DXL IPC functions, see the Rational DOORS API Manual.

Windows programs can also use OLE Automation functions to communicate with other programs.
ipcHostname

**Declaration**

string ipcHostname(string ipAddress)

**Operation**

Resolves the IP address `ipAddress` to its host name.

**Example**

This example prints `localhost` in the DXL Interaction window’s output pane.

```dxl
print ipcHostname("127.0.0.1")
```

server

**Declaration**

IPC server(string socket)

IPC server(int port)

**Operation**

The first form establishes a server connection to the UNIX socket `socket`.

The second form establishes a server connection to the port number `port` on all platforms. In the case that supplied port number is 0, an ephemeral port number is allocated by the operating system. To fetch this ephemeral port number, use `getPort()` on the resulting IPC.

g.getPort

**Declaration**

int getPort(IPC channel)

**Operation**

Fetches the port associated with the specified IPC. Useful when the IPC is allocated an ephemeral port by the operating system (see IPC server(int)).

client

**Declaration**

IPC client(string socket)

IPC client(int ip, string host)
Operation

The first form establishes a client connection to the UNIX socket socket.
The second form establishes a client connection to the IP address ip at host on all platforms.

---

accept

Declaration

bool accept(IPC)

Operation

Waits for a client connection at the server end of the connection.

---

send

Declaration

bool send(IPC chan,
string message)

Operation

Sends the string message down IPC channel chan.

---

recv

Declaration

bool recv(IPC chan,
{string|Buffer} &response
[,]int tmt])

Operation

Waits for a message to arrive in channel chan and assigns it to string or buffer variable response.
The optional third argument defines a timeout, tmt seconds, for a message to arrive in channel chan. If tmt is zero, these functions wait forever. They only work if the caller is connected to the channel as a client or a server.

---

disconnect

Declaration

void disconnect(IPC chan)

Operation

Disconnects channel chan.
delete(IPC channel)

Declaration

void delete(IPC chan)

Operation

Deletes channel chan (can be a server or a client).

System clipboard functions

copyToClipboard

Declaration

bool copyToClipboard(string s)

Operation

Copies a plain text string (not RTF) to the clipboard. On success, returns true.

setRichClip

Declaration

void setRichClip(RTF_string__ s, string styleName, string fontTable)
void setRichClip(Buffer buff, string styleName, string fontTable)
void setRichClip(RTF_string__ s, string styleName, string fontTable, bool keepBullets, bool keepIndents)
void setRichClip(Buffer buff, string styleName, string fontTable, bool keepBullets, bool keepIndents)

Operation

First form sets the system clipboard with the rich text obtained by applying the style styleName to the string s, using the font table fontTable supplied, which should include a default font. Font numbers in the string s will be translated to the supplied font table fontTable.

Second form is same as the first but the source is a buffer buff rather than an RTF_string__.

Third form sets the system clipboard with the rich text obtained by applying the style styleName to the string s, using the font table fontTable supplied. If keepBullets is false, any bullet characters are removed from string s. If keepIndents is false, any indentation is removed from string s. If keepBullets and keepIndents are both true, the behavior is exactly the same as the first form.
Fourth form is same as the third but the source is a buffer `buff` rather than an `RTF_string`.

**Example 1**

The following code:

```dxl
string s = "hello"
string fontTable = "\\deff0{\\fonttbl {\\f1 Times New Roman;}}"
setRichClip(richText s, "Normal", fontTable)
```

puts the following rich text string onto the system clipboard:

```
{\rtf1 \deff0{\fonttbl {\f1 Times New Roman;}}{\stylesheet {\s1 Normal;}}{\s1 hello\par}}
```

**Example 2**

```dxl
string bulletedString = "\\{\\rtf1\\ansi\\ansicpg1252\\deff0\\deflang1033\\fonttbl{\\f0\\fswiss\\fcharset0 Arial;}\\f1\\fnil\\fcharset2 Symbol;}\\viewkind4\\uc1\\pard\\f0\\fs20 Some text with\\par\\pard{\\pntext\\f1\\'B7\tab}{\*\pn\\pnlvlblt\\pnf1\\pnindent0{\\pntxtb\\'B7}}\\fi-720\\li720 bullet 1\\par\\pntext\\f1\\'B7\tab}bullet 2\\par\\pard bullet points in it.\\par\\par}
"
```

```dxl
string fontTable = "\\deff0{\\fonttbl{\\f0\\fswiss\\fcharset0 Arial;}\\f1\\fnil\\fcharset2 Symbol;}
```

setRichClip(richText bulletedString, "Normal", fontTable)

```
// the previous call puts
// "{\\rtf1 \deff0{\\fonttbl{\\f0\\fswiss\\fcharset0 Arial;}\\f1\\fnil\\fcharset2 Symbol;}\\stylesheet {\\s1 Normal;}}{\\s1 Some text with\par {\\f1\\'b7\tab}bullet 1\par {\\f1\\'b7\tab}bullet 2\par bullet points in it.\par \par}}"
```

setRichClip(richText bulletedString, "Normal", fontTable, false, false)

```
// the previous call puts
```
// "{\rtf1 \deff0{\fonttbl{\f0\fswiss\fcharset0 Arial;}{\f1\fnil\fcharset2 Symbol;}}{\stylesheet {\s1 Normal;}}{\s1 Some text with\par bullet 1\par bullet 2\par bullet points in it.\par \par}}"

// on the clipboard -- note no bullet symbols (\'b7) in the markup
Chapter 13  
Customizing Rational DOORS

This chapter explains how you can customize Rational DOORS:

- Color schemes
- Database Explorer options
- Locales
- Codepages
- Message of the day
- Database Properties

Color schemes

This section defines constants and functions for setting the Rational DOORS color scheme.

Display Color Schemes

The following constants are defined as database display schemes for use with the functions below:

originalDOORSColorScheme
modernDOORSColorScheme
highContrastOneColorScheme
highContrastTwoColorScheme
highContrastBlackColorScheme
highContrastWhiteColorScheme

getDefaultColorScheme

Declaration

int getDefaultColorScheme()  

Operation

Returns the default color scheme used by the Database Explorer. The possible values for colorScheme are listed above.
setDefaultColorScheme

**Declaration**

```c
void setDefaultColorScheme(int colorScheme)
```

**Operation**

Sets the default color scheme used by the Database Explorer. Schemes can be created and modified using the **Display** tab in the Options dialog box (from the **Tools > Options** menu in the Database Explorer. The possible values for `colorScheme` are listed above:

optionsExist

**Declaration**

```c
bool optionsExist(string schemeName)
```

**Operation**

Returns `true` if a color scheme exists under `schemeName`; otherwise, returns `false`.

resetColors

**Declaration**

```c
void resetColors([int colorScheme])
```

**Operation**

If no argument is supplied, resets to the default color scheme otherwise resets to `colorScheme`, which can be any of the values listed above.

resetColor

**Declaration**

```c
void resetColor(int colorIndex [,int colorScheme])
```

**Operation**

Resets the color specified by `colorIndex` to the default, or if the second argument is supplied, to `colorScheme`, which can be any of the values listed above.
Database Explorer options

This section defines constants and functions for customizing the Database Explorer.

Font constants

Declaration
int HeadingsFont
int TextFont
int GraphicsFont

Operation
These constants define the font in the `getFontSettings` and `setFontSettings` functions.

getFontSettings

Declaration
void getFontSettings(int level,
int usedIn,
int &size,
int &family,
bool &bold,
bool &italic)

Operation
Passes back settings for the font `usedIn` for objects at heading level `level`. The value of `usedIn` can be `HeadingsFont`, `TextFont`, or `GraphicsFont`. The last four arguments pass back the point size, font family, whether the font is bold, and whether the font is italic. The constants for point size are the following:

- 14pt = 5
- 12pt = 4
- 11pt = 3
- 10pt = 2
- 9pt = 1
- 8pt = 0
setFontSettings

Declaration

void setFontSettings(int level,
          int usedIn,
          int size,
          int family,
          bool bold,
          bool italic)

Operation

Sets the point size, font family, whether the font is bold, and whether the font is italic for the font usedIn for objects at heading level level. The value of usedIn can be HeadingsFont, TextFont, or GraphicsFont. The constants for point size are the following:

• 14pt = 5
• 12pt = 4
• 11pt = 3
• 10pt = 2
• 9pt = 1
• 8pt = 0

refreshExplorer

Declaration

void refreshExplorer(Module m)

Operation

Refreshes the Database Explorer window for module m.

synchExplorer

Declaration

void synchExplorer(Module m)

Operation

Refreshes the Rational DOORS Module Explorer window to reflect changes to the current object selected in the module display.
refreshDBExplorer

Declaration
void refreshDBExplorer()

Operation
Refreshes the Database Explorer window to reflect changes to the current folder or the display state. If the current folder/project is changed using DXL, this perm will not change the currently open item to reflect this. This is used to only refresh the contents of the currently selected item.

setShowFormalModules, setShowDescriptiveModules, setShowLinkModules

Declaration
void setShowFormalModules(bool expression)
void setShowDescriptiveModules(bool expression)
void setShowLinkModules(bool expression)

Operation
Shows formal, descriptive, or link modules in the Database Explorer if expression is true. Hides formal, descriptive, or link modules if expression is false.

showFormalModules, showDescriptiveModules, showLinkModules(get)

Declaration
bool showFormalModules()
bool showDescriptiveModules()
bool showLinkModules()

Operation
Returns true if the Database Explorer is set to show formal, descriptive, or link modules; otherwise returns false.

getSelectedItem

Declaration
Item getSelectedItem()

Operation
Return the item currently selected in the Database Explorer.
Mini database explorer

fnMiniExplorer

Creates a miniature database explorer window that shows a tree view in which you can navigate through the hierarchy of the Rational DOORS database and select an item. The layout of the tree view depends on whether your main client window is configured for Project View or Database View. The items that are displayed can be controlled through the use of a filter.

By default, only projects and folders are populated. You must explicitly choose module types. For ease of use, it is possible to have the tree view expand to a particular start location. The dialog box is modal and therefore prevents other use of the Rational DOORS client while it is displayed.

Declaration

string fnMiniExplorer([Folder f | DB parent,] int itemFilter, string titleBar, string userPrompt)

Operation

Returns the name of the module if found otherwise returns an empty string.

Optional parameter f specifies the folder where the tree-view will auto-expand.
Optional parameter parent specifies the parent DXL dialog.

Parameter titleBar specifies the title window. If no value is specified (i.e. null or empty string) then “DOORS Database Mini-Explorer” will be shown.

Parameter userPrompt specifies the user prompt. If no value is specified (i.e. null or empty string) then "Please make your selection..." will be shown.

Parameter itemFilter specifies what module types are shown in the tree-view. This is a bit mask that can have the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Use this flag to include...</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINI_EXP_LINK_MODS</td>
<td>Link modules</td>
</tr>
<tr>
<td>MINI_EXP_FORMAL_MODS</td>
<td>Formal modules</td>
</tr>
<tr>
<td>MINI_EXP_DESCRIPITIVE_MODS</td>
<td>Descriptive modules</td>
</tr>
</tbody>
</table>
Example

```dxl
string moduleName = fnMiniExplorer(folder("/My Project/My Folder"),
MINI_EXP_FORMAL_MODS | MINI_EXP_LINK_MODS, "Browse", "Select a source module")
```

**Locales**

**getDateFormat**

**Declaration**

```dxl
string getDateFormat([Locale l],[bool isShortFormat])
```

**Operation**

When called with no arguments, this returns the current default short date format. This may be selected for the current user locale, using the Windows Control Panel. If the boolean argument is supplied and is `false`, the default long date format is returned.

**Locale type**

**Operation**

This type represents any valid user locale value. It can take any of the values supported by the client system. The perms that take a Locale argument will all return a DXL run-time error if they are supplied with a null value.

**for Locale in installedLocales**

**Declaration**

```dxl
for Locale in installedLocales
```

**Operation**

This iterator returns all the Locale values installed on the client system.
Example
Locale loc
for loc in installedLocales do
{
    print id(loc) " : " name(loc) " \n"
}

for Locale in supportedLocales

Declaration
for Locale in supportedLocales

Operation
This iterator returns all the Locale values supported on the client system.

userLocale

Declaration
Locale userLocale()

Operation
This returns the current user locale on the client system.

name

Declaration
string name(Locale l)

Operation
This returns the name (in the current desktop language) of the specified Locale.

language

Declaration
string language(Locale l)

Operation
This returns the English name of the Locale language.
region

**Declaration**

string region(Locale l)

**Operation**

This returns the English name of the country/region of the Locale.

id

**Declaration**

int id(Locale l)

**Operation**

This returns the integer identifier value for the Locale. This is a constant for any given Locale.

locale

**Declaration**

Locale locale(int i)

**Operation**

This returns the Locale for the specified identifier value. It returns null if the integer value is not a valid supported locale identifier.

installed

**Declaration**

bool installed(Locale l)

**Operation**

This returns true if the Locale is installed on the client machine. Otherwise it returns false.

attributeValue

**Declaration**

bool attributeValue(AttrDef attr, string s[, bool bl])
Operation
Tests whether the supplied string represents a valid value for the specified attribute definition. If the third argument is supplied and set to true, the function will return true if the attribute base type is date and the string is a valid date string for the user’s current Locale setting.

locale

Declaration
AttrDef.locale()

Operation
Use to access the locale of the specified AttrDef. It returns null if there is no locale specified by the attribute definition.

Example
AttrDef ad = find(current Module, "Object Text")
Locale loc = ad.locale
print "Object Text locale is " name(loc) "\n"

getLegacyLocale

Declaration
Locale getLegacyLocale(void)

Operation
Returns the legacy data locale setting for the database. This determines the locale settings that are used to display legacy attribute data. If none is set, this returns null, and legacy attribute values are displayed according to the settings for the current user locale.

setLegacyLocale

Declaration
string setLegacyLocale(Locale l)

Operation
This enables users with Manage Database privilege to set the Legacy data locale for the database (as explained above). setLegacyLocale(null) removes the Legacy data locale setting for the database. Returns null on success, and an error string on failure, including when it is called by a user without Manage Database privilege.
Single line spacing constant

Declaration
int single

Operation
This constant is used to specify single line spacing.

Line spacing constant for 1.5 lines

Declaration
int onePointFive

Operation
This constant is used to specify 1.5 lines line spacing.

setLineSpacing

Declaration
void setLineSpacing(int lineSpacing)

Operation
Sets line spacing for the current locale.

Example
setLineSpacing(single)

getLineSpacing

Declaration
int getLineSpacing()

Operation
Retrieves the line spacing for the current locale.

Example
if (getLineSpacing() == onePointFive)
{
    print "Line spacing is set to One and a half lines.\n"}
setLineSpacing

Declaration
void setLineSpacing(Locale locale, int lineSpacing)

Operation
Sets line spacing for the desired locale.

getLineSpacing

Declaration
int getLineSpacing(Locale locale)

Operation
Retrieves the line spacing for the desired locale.

getDefaultLineSpacing

Declaration
int getDefaultLineSpacing( void)

Operation
Returns the default line spacing for the user’s current locale. For example, it will return single when the line spacing is European, onePointFive when the line spacing is Japanese, Chinese, or Korean, and so on.

defFontSettings

Declaration
void getFontSettings(int level, int usedIn, int &size, string &family, bool &bold, bool &italic, Locale locale)

Operation
Gets the current user’s font-related display options for the locale provided. The usedIn parameter can be one of the following constants: HeadingsFont, TextFont or GraphicsFont.

Example
int pointSize
string fontFamily
bool bold, italic
getFontSettings(2, TextFont, pointSize, fontFamily, bold, italic, userLocale)
print fontFamily ", " pointSize ", " bold ", " italic "\n"

setFontSettings

Declaration
void setFontSettings(int level, int usedIn, int size, string family, bool bold,
bool italics, Locale locale)

Operation
Sets the current user's font-related display options for the locale provided.

for string in availableFonts do

Declaration
for string in availableFonts do {}

Operation
Iterator over the specified availableFonts.

Example
string fontName
for fontName in availableFonts do {
  ...
}
Provides access to the names of each of the available fonts.

Codepages

Constants

Constants for codepages
The following constants denote codepages:
•   const int CP_LATIN1 // ANSI Latin-1
•   const int CP_UTF8 // Unicode UTF-8 encoding
• const int CP_UNICODE // UTF-16 little-endian encoding (= CP_UTF16_LE)
• const int CP_UTF16_LE // UTF-16 little-endian encoding
• const int CP_UTF16_BE // UTF-16 big-endian encoding
• const int CP_JAP // Japanese (Shift-JIS)
• const int CP_CHS // Simplified Chinese (GB2312)
• const int CP_KOR // Korean (KSC 5601)
• const int CP_CHT // Traditional Chinese (Big 5)

for int in installedCodepages

    Declaration
    for int in installedCodepages do
    Operation
    This iterator returns the values of all the codepages installed in the client system.

for int in supportedCodepages

    Declaration
    for int in supportedCodepages do
    Operation
    This iterator returns the values of all codepages supported by the client system. Some of these may not be currently installed.

currentANSIcodepage

    Declaration
    int currentANSIcodepage()
    Operation
    Returns the current default ANSI codepage for the client system. For example, in Western Europe and North America this will typically return 1252, equivalent to ANSI Latin-1.

codepageName

    Declaration
    string codepageName(int codepage)
Operation
This returns the name of the specified codepage. Note that this returns an empty string for any codepage that is not installed on the system.

---

read

Declaration
Stream read(string filename, int codepage)

Operation
Opens a stream onto the specified filename; content of file decoded from the specified codepage.

---

write

Declaration
Stream write(string filename, int codepage)

Operation
Opens a stream onto the specified filename; content of file encoded to the specified codepage.

---

append

Declaration
Stream append(string filename, int codepage)

Operation
Opens a stream for append onto the specified filename; content of file encoded to the specified codepage.

---

readFile

Declaration
string readFile(string filename, int codepage)

Operation
Reads string from specified file; content is decoded from the specified codepage.

Note: The Files function also has a readFile operator. For information about Files and readFile, see “readFile,” on page 136.
isValidChar

Declaration

bool isValidChar(char c, int codepage)

Operation

Returns true only if the supplied character can be represented in the specified codepage.

convertToCodepage

Declaration

{string|Buffer} convertToCodepage(int codepage, {string|Buffer&} utf8string)

Operation

Returns a version of the supplied string or buffer, encoded according to the specified codepage. The supplied string is assumed to be encoded in UTF-8 (the default encoding for all Rational DOORS strings).

Note: Only UTF-8 strings will print and display correctly in Rational DOORS V8.0 and higher. This perm is intended for use in exporting string data for use in other applications.

Example

string latin1str = covertToCodepage(CP_LATIN1, “für Elise”)

convertFromCodepage

Declaration

{string|Buffer} convertFromCodepage(int codepage, {string|Buffer&} cpString)

Operation

Converts a string or buffer from the specified codepage to the Rational DOORS default UTF-8 encoding. Once a non-UTF-8 string is converted to UTF-8, it can be displayed and printed by Rational DOORS, including 8-bit (non-ASCII) characters.

Example

int port=5093
int iTimeOut=10
IPC ipcServerConn=server(port)
string inputStr

if (!accept(ipcServerConn))

{
    print "No connection\n";
}
else while (recv (ipcServerConn, inputStr, iTimeOut))
{
    inputStr = convertFromCodepage(currentANSIcodepage(), inputStr)
    print inputStr "\n";
}

---

**Message of the day**

**setMessageOfTheDay**

**Declaration**

string setMessageOfTheDay(string message)

**Operation**

This is used to set the message of the text in the database. Returns null if successful, returns an error if the user does not have the manage database privilege.

**setMessageOfTheDayOption**

**Declaration**

string setMessageOfTheDayOption(bool setting)

**Operation**

Used to turn the message of the day on or off. Returns an error if the user does not have the manage database privilege, otherwise returns null.

**getMessageOfTheDay**

**Declaration**

string getMessageOfTheDay()

**Operation**

Returns the message of the day if one is set, otherwise returns null.
getMessageOfTheDayOption

Declaration

bool getMessageOfTheDayOption()

Operation

Used to determine whether the message of the day is enabled. Returns true if it is enabled, otherwise returns false.

Example

string s1, s2, message
message = "Hello and welcome to DOORS!"

if (getMessageOfTheDayOption()){
    print "Current message of the day is : " (getMessageOfTheDay())
} else {
    print "No message of the day is set, setting message and turning on." 
    s1 = setMessageOfTheDay(message)
    if (!null s1){
        print "There was an error setting the message of the day : " s1
    } else {
        s2 = setMessageOfTheDayOption(true)
        if (!null s2){
            print "There was an error turning on the message of the day :" s2
        }
    }
}

Database Properties

setLoginFailureText

Declaration

string setLoginFailureText(string msg)
Operation
Sets the string as the pretext for login failure Emails sent through Rational DOORS. Returns null on success or failure error message.

getLoginFailureText

Declaration
string getLoginFailureText(void)

Operation
Gets the string used for login failure Emails sent through Rational DOORS.

setDatabaseMailPrefixText

Declaration
string setDatabaseMailPrefixText(string msg)

Operation
Sets the string as the pretext for Emails sent through Rational DOORS. Returns null on success or failure error message.

getDatabaseMailPrefixText

Declaration
string getDatabaseMailPrefixText(void)

Operation
Gets the string used in Emails sent through Rational DOORS.

setEditDXLControlled

Declaration
string setEditDXLControlled(bool)

Operation
Activates or de-activates the database wide setting determining whether the ability to edit DXL will be controlled. Returns null on success, or an error on failure.
getEditDXLControlled

Declaration

bool getEditDXLControlled(void)

Operation

Used to determine if the ability to edit DXL is controlled in the database. Returns true if the ability to edit DXL can be denied.
Chapter 14
Rational DOORS database access

This chapter covers:
• Database properties
• Group and user manipulation
• Group and user management
• LDAP
• LDAP Configuration
• LDAP server information
• LDAP data configuration
• Rational Directory Server

Database properties

This section defines functions for Rational DOORS database properties. DXL defines the data type LoginPolicy, which can take either of the following values:
viaDOORSLogin
viaSystemLogin
These values control how users log in to Rational DOORS, using the Rational DOORS user name or the system login name.

getDatabaseName

Declaration
string getDatabaseName()

Operation
Returns the name of the Rational DOORS database.

setDatabaseName

Declaration
bool setDatabaseName(string newName)
Operation
Sets the name of the Rational DOORS database to newName. If the operation succeeds, it returns true; otherwise, it returns false. The operation fails if the name contains any prohibited characters.
This perm only operates if the current user has the Manage Database privilege, otherwise it returns false.

getAccountsDisabled

Declaration
bool getAccountsDisabled()

Operation
If standard and custom user accounts for the current database are disabled, returns true; otherwise, returns false.

Example
if (getAccountsDisabled()) {
  print "Only those with May Manage Power can log in"
}

setAccountsDisabled

Declaration
void setAccountsDisabled(bool disabled)

Operation
Disables or enables standard and custom user accounts for the current database, depending on the value of disabled.
This perm only operates if the current user has the Manage Database privilege, otherwise an error message is displayed.

Note: A saveDirectory() command must be used for this to take effect.

Example
This example disables all standard and custom user accounts:
setAccountsDisabled(false)
saveDirectory()

getDatabaseIdentifier

Declaration
string getDatabaseIdentifier()
Operation
Returns the unique database identifier generated by Rational DOORS during database creation.

getDatabasePasswordRequired

Declaration
bool getDatabasePasswordRequired()

Operation
Returns true if passwords are required for the current Rational DOORS database; otherwise, returns false.

setDatabasePasswordRequired

Declaration
void setDatabasePasswordRequired(bool required)

Operation
Sets passwords required or not required for the current database, depending on the value of required. This perm only operates if the current user is the administrator, otherwise an error message is displayed.

getReconfirmPasswordRequired

Declaration
bool getReconfirmPasswordRequired()

Operation
Returns true if a reconfirmation password is required after a specified timeout period; otherwise, returns false.

setReconfirmPasswordRequired

Declaration
void setReconfirmPasswordRequired(bool required)

Operation
Sets whether a reconfirmation password is required after a specified timeout period, depending on the value of required. This perm only operates if the current user has the Manage Database privilege.
getReconfirmPasswordTimeout

Declaration
int getReconfirmPasswordTimeout()

Operation
Returns the timeout period (in minutes) before the reconfirmation password dialog appears.

setReconfirmPasswordTimeout

Declaration
void setReconfirmPasswordTimeout(int timeout)

Operation
Sets the timeout period to timeout minutes before the reconfirmation password dialog appears.
This perm only operates if the current user has the Manage Database privilege.

getRequireLettersInPassword

Declaration
bool getRequireLettersInPassword()

Operation
Returns true if a password is required to contain at least one alphabetic character; otherwise, returns false.

setRequireLettersInPassword

Declaration
string setRequireLettersInPassword(bool required)

Operation
If required is true, then a password is required to contain at least one alphabetic character.
This perm only operates if the current user has the Manage Database privilege, otherwise an error message is returned.

getRequireNumberInPassword

Declaration
bool getRequireNumberInPassword()
Operation

Returns true if a password is required to contain at least one number; otherwise, returns false.

setRequireNumberInPassword

Declaration

string setRequireNumberInPassword(bool required)

Operation

If required is true, a password is required to contain at least one number.

This perm only operates if the current user has the Manage Database privilege, otherwise an error message is returned.

getRequireSymbolInPassword

Declaration

bool getRequireSymbolInPassword()

Operation

Returns true if a password is required to contain at least one non-alphanumeric character; otherwise, returns false.

setRequireSymbolInPassword

Declaration

string setRequireSymbolInPassword(bool required)

Operation

If required is true, a password is required to contain at least one non-alphanumeric character.

This perm only operates if the current user has the Manage Database privilege, otherwise an error message is returned.

getDatabaseMinimumPasswordLength

Declaration

int getDatabaseMinimumPasswordLength()

Operation

Returns the minimum number of characters required for a password on the current database.
setDatabaseMinimumPasswordLength

**Declaration**

```c
void setDatabaseMinimumPasswordLength(int length)
```

**Operation**

Sets the length of password required for the current database to `length` characters. The value can be any non-negative integer.

This perm only operates if the current user has the Manage Database privilege.

getMinPasswordGeneration

**Declaration**

```c
int getMinPasswordGeneration()
```

**Operation**

Returns the minimum number of password generations before a password can be reused.

setMinPasswordGeneration

**Declaration**

```c
string setMinPasswordGeneration(int num)
```

**Operation**

Sets the minimum number of password generations before a password can be reused to `num`. The minimum number cannot exceed the in-built maximum limit of 12 generations before a password can be reused.

This perm only operates if the current user has the Manage Database privilege, otherwise an error message is returned.

getMaxPasswordGenerationLimit

**Declaration**

```c
int getMaxPasswordGenerationLimit()
```

**Operation**

Returns the in-built maximum limit of password generations before a password can be reused. This maximum limit is set to 12.
getMinPasswordAgeInDays

Declaration
int getMinPasswordAgeInDays()

Operation
Returns the minimum number of days before a password can be reused.

setMinPasswordAgeInDays

Declaration
string setMinPasswordAgeInDays(int days)

Operation
Sets the minimum number of days before a password can be reused to days. The minimum number cannot exceed the in-built maximum limit of 180 days before a password can be reused.
This perm only operates if the current user has the Manage Database privilege, otherwise an error message is returned.

getMaxPasswordAgeLimit

Declaration
int getMaxPasswordAgeLimit()

Operation
Returns the in-built maximum limit of days before a password can be reused. This maximum limit is set to 180 days.

getDatabaseMailServer

Declaration
string getDatabaseMailServer(void)

Operation
Returns as a string the name of the SMTP mail server for Rational DOORS.

setDatabaseMailServer

Declaration
void setDatabaseMailServer(string serverName)
Operation
Sets the mail server for the current database to `servername`.
This perm only operates if the current user has the Manage Database privilege.

getDatabaseMailServerAccount

Declaration
`string getDatabaseMailServerAccount(void)`

Operation
Returns as a string the name of the mail account that appears to originate messages from Rational DOORS.

setDatabaseMailServerAccount

Declaration
`void setDatabaseMailServerAccount(string accountName)`

Operation
Sets to `accountName` the mail account that appears to originate messages from Rational DOORS.
This perm only operates if the current user has the Manage Database privilege.

getLoginPolicy

Declaration
`LoginPolicy getLoginPolicy()`

Operation
Returns the login policy (either `viaDOORSLogin` or `viaSystemLogin`) for the current database. These values control how users log in to Rational DOORS, using the Rational DOORS name or the system login name.

setLoginPolicy

Declaration
`void setLoginPolicy(LoginPolicy policy)`

Operation
Sets the login policy for the current database to `policy`, which can be either `viaDOORSLogin` or `viaSystemLogin`.
This perm only operates if the current user has the Manage Database privilege, otherwise an error message is displayed.
getDisableLoginThreshold

Declaration

int getDisableLoginThreshold()

Operation

Returns the number of times a user account tolerates a failed login. If the number of login failures to any single account exceeds this value, Rational DOORS disables that account. Nobody can use a disabled account.

If the return value is zero, there is no limit. See also the getFailedLoginThreshold function.

setDisableLoginThreshold

Declaration

void setDisableLoginThreshold(int attempts)

Operation

Sets the number of times a user account tolerates a failed login. If the number of login failures to any single account exceeds this value, Rational DOORS disables that account. Nobody can use a disabled account.

If attempts is zero, there is no limit. See also the setFailedLoginThreshold function.

This perm only operates if the current user has the Manage Database privilege, otherwise an error message is displayed.

getFailedLoginThreshold

Declaration

int getFailedLoginThreshold()

Operation

Returns the number of times Rational DOORS tolerates a login failure. If this threshold is exceeded, Rational DOORS closes.

If the return value is zero, there is no limit. See also the setDisableLoginThreshold function.

setFailedLoginThreshold

Declaration

void setFailedLoginThreshold(int attempts)

Operation

Sets the number of times Rational DOORS tolerates a login failure. If this threshold is exceeded, Rational DOORS closes.

If attempts is zero, there is no limit. See also the setDisableLoginThreshold function.
This perm only operates if the current user has the Manage Database privilege, otherwise an error message is displayed.

**Note:** A `saveDirectory()` command must be used for this to take effect.

**Example**

```dxl
setFailedLoginThreshold(3)
saveDirectory()
```

---

### getLoginLoggingPolicy

**Declaration**

```dxl
bool getLoginLoggingPolicy(bool type)
```

**Operation**

If Rational DOORS is keeping track of logins of the specified type, returns `true`; otherwise, returns `false`. If type is `true`, returns the policy for successful logins; otherwise, returns the policy for login failures.

To set the logging policy, use the `setLoginLoggingPolicy` function.

**Example**

This example indicates whether Rational DOORS is keeping track of login failures.

```dxl
getLoginLoggingPolicy(false)
```

---

### setLoginLoggingPolicy

**Declaration**

```dxl
void setLoginLoggingPolicy(bool type, bool status)
```

**Operation**

Sets the logging policy for login events of the specified type. If `status` is `true`, logging of the specified type is enabled; otherwise, it is disabled. If `type` is `true`, sets the policy for successful logins; otherwise, sets the policy for login failures.

To find out the current logging policy, use the `getLoginLoggingPolicy` function.

**Example**

This example causes Rational DOORS not to log successful logins.

```dxl
setLoginLoggingPolicy(true, false)
```

---

### setMinClientVersion

**Declaration**

```dxl
string setMinClientVersion(string s)
```
Operation

Sets the minimum client version that can connect to the current database. The string argument must be of the format \( n.n \), \( n.n.n \) or \( n.n.n.n \), where each \( n \) is a decimal integer. The integer values represent Major version, Minor version, Service Release and Patch number respectively. The Service Release and Patch numbers are optional, and default to zero.

This perm only operates if the current user has the Manage Database privilege, otherwise it returns an appropriate error string. It also returns an error string if the string argument is not of the correct format, or represents a client version higher than the current client.

getMinClientVersion

Declaration

string getMinClientVersion(void)

Operation

Returns a string representing the minimum client version that can connect to the current database, in the format \( n.n \), \( n.n.n \) or \( n.n.n.n \). The format is explained in setMinClientVersion. If no minimum client version has been set for the database, this perm returns a NULL string.

setMaxClientVersion

Declaration

string setMaxClientVersion(string s)

Operation

Sets the maximum client version that can connect to the current database. The string argument must be of the format \( n.n \), \( n.n.n \) or \( n.n.n.n \), where each \( n \) is a decimal integer. The integer values represent Major version, Minor version, Service Release and Patch number respectively. The Service Release and Patch numbers are optional.

This perm only operates if the current user has the Manage Database privilege, otherwise it returns an appropriate error string. It also returns an error string if the string argument is not of the correct format, or represents a client version lower than the current client.

getMaxClientVersion

Declaration

string getMaxClientVersion(void)

Operation

Returns a string representing the maximum client version that can connect to the current database, in the format \( n.n \), \( n.n.n \) or \( n.n.n.n \). The format is explained in setMinClientVersion. If no minimum client version has been set for the database, this perm returns a null string.
doorsInfo

Declaration
string doorsInfo(int i)

Operation
A new valid value for the integer argument is defined (infoServerVersion).
This returns the version of the database server to which the client is currently connected.

Example
string serverVersion = doorsInfo(infoServerVersion)
print "database server version is " serverVersion "\n"

addNotifyUser

Declaration
void addNotifyUser(User user)

Operation
Adds user to the list of users to be notified by e-mail of attempts to log in. If user does not have an e-mail address, no notification takes place.

deleteNotifyUser

Declaration
void deleteNotifyUser(User user)

Operation
Deletes user from the list of users to be notified by e-mail of attempts to log in.

createPasswordDialog

Declaration
string createPasswordDialog(DB parent,
                                bool &completed)

Operation
Displays a dialog box containing password and password confirmation fields as well as OK and Cancel buttons. The parent argument is needed for the Z-order of the elements.
If confirmation is successful, returns a null string; otherwise, returns an error message.
If the user clicks **OK**, sets *completed* to **true**. If the user clicks **Cancel**, sets *completed* to **false**. Rational DOORS stores the entered password temporarily for the next user account created with the `addUser` function. It is not stored as plain text, and is lost if Rational DOORS shuts down before a new account is created.

**Example**

See the section “Creating a user account example,” on page 228.

---

### changePasswordDialog

**Declaration**

```dxl
define string changePasswordDialog
  returnstring parent,
  User user,
  bool masquerade,
  bool &completed
```

**Operation**

Displays a dialog box containing password and password confirmation fields as well as **OK** and **Cancel** buttons. The *parent* argument is needed for the Z-order of the elements.

If confirmation is successful, returns a null string; otherwise, returns an error message.

If the user clicks **OK**, sets *completed* to **true**. If the user clicks **Cancel**, sets *completed* to **false**. Rational DOORS stores the entered password temporarily. It is not stored as plain text, and is lost if Rational DOORS shuts down before the password is copied using the `copyPassword` function.

A user without the `mayEditUserList` power must confirm his existing password, otherwise the function returns an error message. A user with this power is not prompted for an existing password, unless `masquerade` is **true**.

**Example**

This example copies a new password to the user account for which it was created.

```dxl
User u = find("John Smith")
bool completed
string s = changePasswordDialog(confirm, u,
  false, completed)
if completed && (null s)
  copyPassword()
}
saveUserRecord(u)
saveDirectory()
```
confirmPasswordDialog

Declaration

bool confirmPasswordDialog(DB parent,
   bool &completed)

Operation

Displays a dialog box containing a password confirmation field as well as OK and Cancel buttons. The title of the dialog box is always Confirm password - DOORS. The parent argument is needed for the Z-order of the elements.

If confirmation is successful, returns true; otherwise, returns false.

If the user clicks OK, sets completed to true. If the user clicks Cancel, sets completed to false.

Example

bool bPasswordOK = false, bCompleted = false
// query user
bPasswordOK = confirmPasswordDialog(dbExplorer, bCompleted)
// check status
if (bCompleted == true)
{
   print "Confirmed"
}

copyPassword

Declaration

bool copyPassword()

Operation

Copies the password created using the function to the account for which the password was created. Returns null on success and an error message on failure.

Example

This example copies a new password to the user account for which it was created.

User u = find("John Smith")
bool completed
string s = changePasswordDialog(dbExplorer, u, false, completed)
if (completed && (null s)) {
   copyPassword()
}
getAdministratorName

Declaration

string getAdministratorName()

Operation

Returns the name of the administrator for the Rational DOORS database.

sendEMailNotification

Declaration

{bool|string} sendEMailNotification(string fromDescription,
                              string targetAddress,
                              string subject,
                              string message)

string sendEMailNotification(string fromDescription,
                              Skip targetAddresses,
                              [, Skip ccAddresses]
                              [, Skip bccAddresses]
                              string subject,
                              string message)

Operation

Issues a notification e-mail to the specified address or addresses. The communication takes place using SMTP, and depends on the appropriate Database Properties fields being correctly set up prior to its use (SMTP Mail Server and Mail Account).

The user can set the description of the sender, the subject matter, and message contents using fromDescription, subject and message. If fromDescription is a null string, Rational DOORS defaults to a standard text:

DOORS Mail Server

The following standard text is sent in front of the specified message:

The following is a notification message from DOORS - please do not reply as it was sent from an unattended mailbox.

The variant returning a boolean is for legacy use and returns true if the SMTP communication was successful; otherwise, returns false. Others variants return an error string on failure.
sendEMailMessage

Declaration

{bool|string} sendEMailMessage(
    string fromDescription,
    string targetAddress,
    string subject,
    string message)

string sendEMailMessage(
    string fromDescription,
    Skip targetAddress,
    [, Skip ccAddresses]
    [, Skip bccAddresses]
    string subject,
    string message)

Operation

Performs the same function as sendEMailNotification, but without prepending text to the message.

Creating a user account example

This example creates a new user account named John Smith, having johns as its login name, with whatever password is entered in the dialog box.

// prevent dxl timeout dialog
pragma runLim, 0

// globals
bool g_bPasswordOK = true

// user details
const string sUserName = "John Smith"
const string sUserLogin = "johns"

// only relevant if password is required
if (getDatabasePasswordRequired() == true) {
    bool bConfirmCompleted = false
    // query user
    g_bPasswordOK =
        confirmPasswordDialog(dbExplorer,
            bConfirmCompleted)
    // check status
if (bConfirmCompleted == false) {
    // adjust accordingly
    g_bPasswordOK = false
}

// check status
if (g_bPasswordOK == true) {
    // only relevant if name doesn't exist
    // as group or user
    if (existsUser(sUserName) == false &&
        existsGroup(sUserName) == false) {
        bool bCreateCompleted = false
        // query user
        string sErrorMsg =
            createPasswordDialog(dbExplorer,
                                bCreateCompleted)
        // check status
        if (sErrorMsg == null &&
            bCreateCompleted == true) {
            // add new user
            if (addUser(sUserName, sUserLogin) ==
                null) {
                // save new user list
                if (saveDirectory() == null) {
                    // refresh
                    if (loadDirectory() == null) {
                        // inform user
                        infoBox("User '{sUserName}'
                                was added successfully.\n")
                    } else {
                        // warn user
                        warningBox("Failed to load
                                    user list.\n")
                    }
                } else {
                    // warn user
                    warningBox("Failed to save
                             user list.\n")
                }
            } else {
                // warn user
                warningBox("Failed to add user
                        '{sUserName}'.\n")
            }
        }
    }
} else {
    // warn user
    warningBox("Failed to load
              user list.\n")
} else {
    // warn user
    warningBox("Failed to save
            user list.\n")
}
Group and user manipulation

Group and user manipulation functions and for loops use the following DXL data types: Group, User, GroupList, UserList, and UserNotifyList. These types have the following permitted values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupList</td>
<td>groupList</td>
<td>Provides access to all groups defined in the database. This is the only constant of type GroupList.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UserList</td>
<td>userList</td>
<td>Provides access to all users (with the exception of the administrator account) who have an account in the database. This is the only constant of type UserList.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UserNotifyList</td>
<td>userNotifyList</td>
<td>Provides access to all users who must be notified by e-mail of attempts to log in. This is the only constant of type UserNotifyList.</td>
</tr>
</tbody>
</table>

find

Declaration

User find()
{Group|User} find(string name)

Operation

The first form returns a handle of type User to the currently logged in user.
The second form returns a handle of type Group or type User for the group or user name. A call to this function where name does not exist causes a DXL run-time error. To check that a user or group exists, use the existsGroup, existsUser functions.

**findByID**

**Declaration**

User findById(string identifier)

**Operation**

Returns a handle of type User for the specified identifier, or null if the user does not exist but the identifier is valid. If the specified identifier is badly formed, a DXL run-time error occurs.

You can extract the identifier for a user from a variable of type User with the identifier property (see “Group and user properties,” on page 239).

**existsGroup, existsUser**

**Declaration**

bool existsGroup(string name)

bool existsUser(string name)

**Operation**

If the named group or user exists, returns true; otherwise, returns false.

**loadUserRecord**

**Declaration**

string loadUserRecord(User user)

**Operation**

Loads the details of user user from the database.

**Example**

User u = find("boss")
loadUserRecord(u)
string e = u.email
print e
**ensureUserRecordLoaded**

**Declaration**

```dxl
string ensureUserRecordLoaded(User user)
```

**Operation**

If the user’s record for user has not already been loaded, calls the `loadUserRecord` function.

**saveUserRecord**

**Declaration**

```dxl
string saveUserRecord(User user)
```

**Operation**

Saves the details of user `user` to the database.

**Note:** A `saveDirectory()` command should be used to commit the changes to the database

**Example**

```dxl
User u = find("boss")
loadUserRecord(u)
string e = u.email
if (null e) {
  u.email = "boss@work"
}
saveUserRecord (u)
saveDirectory()
```

**loadDirectory**

**Declaration**

```dxl
string loadDirectory()
```

**Operation**

Loads the group and user list from the database. All changes made since the last load or save are lost. If the operation succeeds, returns null; otherwise, returns an error message.
saveDirectory

Declaration

    string saveDirectory()

Operation

Saves all changes to groups, users, and login policies in the database. If the call fails, returns an error message.

Note: This perm places a temporary lock on the users directory. If used in a continuous manner, for example, repeatedly in a for loop, this could cause conflicts for another user trying to login.

for user in database

Syntax

    for user in userList do {
        ...
    }

where:

user is a variable of type User

If the database is configured to use an LDAP directory, use:

    for user in userList("pattern") do {
        ...
    }

Operation

Assigns the variable user to be each successive non-administrator user in the database.

For LDAP, if the pattern specified is *, then the loop returns the entire set of users that are available in the LDAP directory. This operation might require some time, depending on the number of users in the LDAP directory.

Example

This example prints a list of users in the database:

    User user
    for user in userList("*") do {
        string uName = user.name
        print uName "\n"
    }
for group in database

Syntax

for group in groupList do {
  ...
}

where:

- `group` is a variable of type `Group`

If the database is configured to use an LDAP directory, use:

for group in groupList("pattern") do {
  ...
}

Operation

Assigns the variable `group` to be each successive group in the database.

For LDAP, if the pattern specified is ", then the loop returns the entire set of groups that are available in the LDAP directory. This operation might require some time, depending on the number of groups in the LDAP directory.

Example

This example prints a list of groups in the database:

```dxl
Group group
for group in groupList("*") do {
  string gName = group.name
  print gName "\n"
}
```

for user in group

Syntax

for user in group do {
  ...
}

where:

- `user` is a variable of type `User`
- `group` is a variable of type `Group`
Operation
Assigns the variable \texttt{user} to be each successive non-administrator user in the specified group.

Example
This example prints a list of users in group \texttt{development}:

\begin{verbatim}
User user
Group development = find("development")
for user in development do {
    string uName = user.name
    print uName \"\n\n
\end{verbatim}

\underbrace{\textbf{for group in ldapGroupsForUser}}

Declaration
\begin{verbatim}
for g in ldapGroupsForUser(u) do {
    ...}
\end{verbatim}
where:
\begin{itemize}
    \item \texttt{g} is a variable of type \texttt{Group}
    \item \texttt{u} is a variable of type \texttt{User}
\end{itemize}

Operation
Iterate over all groups of which the user passed to the \texttt{ldapGroupsForUser} function is a member. Note that this iterator is only effective when Rational DOORS is configured for LDAP, not for the Rational Directory Server.

Example
\begin{verbatim}
User u = find("fred")
Group g
for g in ldapGroupsforUser(u) do {
    ...}
\end{verbatim}

\underbrace{\textbf{for user in notify list}}

Syntax
\begin{verbatim}
for user in userNotifyList do {
    ...}
\end{verbatim}
where:

user is a variable of type User

**Operation**

Assigns the variable user to be each successive user in the list of users to be notified by e-mail of login activity.

---

**copyPassword**

**Declaration**

string copyPassword()

**Operation**

This is the same as the existing copyPassword() perm. It performs an identical operation, transferring the shadow password to the real password but instead of returning a boolean indicating success or failure, it returns NULL on success and a message on failure. The existing perm can fail resulting in a reported error in the DXL output display if an exception is thrown. The new perm will catch exceptions and pass the message back to the DXL code for it to display as a pop-up dialog.

---

**fullName**

**Declaration**

UserElement_ fullName()

**Operation**

This can be used to get the full name of the user.

**Example**

User u = find()
string name = u.fullName

---

**mayEditDXL**

**Declaration**

UserElement_ mayEditDXL()

**Operation**

Indicates whether the specified user is able to edit and run DXL programs.

**Example**

User u = find
bool useDXL = u.mayEditDXL
synergyUsername

**Declaration**

UserElement_ synergyUsername()

**Operation**

This can be used to retrieve the user's SYNERGY/Change user name.

This attribute value is only available when Rational DOORS is configured to use the Rational Directory Server.

This value is not writable; its value is set when the systemLoginName is set.

**Example:**

```dxl
User u = find("Test")
string s = u.synergyUsername
User u = find("Test")
u.synergyUsername = "testuser"
//this generates an error
```

tForename

**Declaration**

UserElement_ forename()

**Operation**

This can be used to get or set the user’s forename.

This attribute value is only available when Rational DOORS is configured to use the Rational Directory Server.

Setting this value has the side effect of setting the fullName of the user to the concatenation of forename and surname. This is only relevant when configured to use the Rational Directory Server.

**Example**

```dxl
User u = find("Test")
string s = u.forename

User u = find("Test")
u.forename = "Tom"
```
surname

Declaration

UserElement_ surname()

Operation

This can be used to get or set the user’s surname.

This attribute value is only available when Rational DOORS is configured to use the Rational Directory Server.

Setting this value has the side effect of setting the fullName of the user to the concatenation of forename and surname. This is only relevant when configured to use the Rational Directory Server.

Example

User u = find("Test")
string s = u.surname

User u = find("Test")
u.surname = "Thumb"

Group and user management

Group and user management functions use the DXL data types Group, User, and UserClass.

User class constants

Type UserClass can have one of the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>administrator</td>
<td>User type administrator</td>
</tr>
<tr>
<td>standard</td>
<td>User type standard</td>
</tr>
<tr>
<td>databaseManager</td>
<td>User type database manager</td>
</tr>
<tr>
<td>projectManager</td>
<td>User type project manager</td>
</tr>
<tr>
<td>custom</td>
<td>User type custom</td>
</tr>
</tbody>
</table>
## Group and user properties

Properties are defined for use with the . (dot) operator and a group or user handle to extract information from, or specify information in a group or user record, as shown in the following syntax:

```
variable.property
```

where:

- `variable` is a variable of type `Group` or `User`
- `property` is one of the user or group properties

The following tables list the group properties and the information they extract or specify (for further details on specifying information see the `setGroup` function):

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boolean property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disabled</td>
<td>whether the group is disabled</td>
</tr>
</tbody>
</table>

The following tables list the user properties and the information that they extract or specify.

**Note:** The string properties and Boolean properties in the following tables do not apply to the following DXL statements. These statements only use one property, the Boolean property `Disabled`:

- for property in user account
- `isAttribute(user)`
- `delete(user property)`
- `get(user property)`
- `set(user property)`

For further details on specifying information, see the `setUser` function.

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>postal address</td>
</tr>
<tr>
<td>email</td>
<td>e-mail address</td>
</tr>
<tr>
<td>identifier</td>
<td>identifier: a string containing a hexadecimal number, which is created by Rational DOORS</td>
</tr>
<tr>
<td>description</td>
<td>description</td>
</tr>
</tbody>
</table>
### String property

<table>
<thead>
<tr>
<th>Property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>password</td>
<td>password (write-only)</td>
</tr>
<tr>
<td>systemLoginName</td>
<td>system login name (not Rational DOORS user name)</td>
</tr>
<tr>
<td>telephone</td>
<td>telephone number</td>
</tr>
<tr>
<td>fullName</td>
<td>full name</td>
</tr>
</tbody>
</table>

### Boolean property

<table>
<thead>
<tr>
<th>Property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disabled</td>
<td>whether the account is disabled</td>
</tr>
<tr>
<td>emailCPUpdates</td>
<td>whether the user of the CP system can be notified by e-mail when the status of a proposal changes, for example when it is accepted or rejected</td>
</tr>
<tr>
<td>mayArchive</td>
<td>whether the user can archive and restore modules and projects</td>
</tr>
<tr>
<td>mayCreateTopLevelFolders</td>
<td>whether the user can create folders at the root of the database</td>
</tr>
<tr>
<td>mayEditGroupList</td>
<td>whether the user can edit, create and delete groups</td>
</tr>
<tr>
<td>mayEditUserList</td>
<td>whether the user can edit, create, and delete user accounts and groups</td>
</tr>
<tr>
<td>mayManage</td>
<td>whether the user can manage the Rational DOORS database</td>
</tr>
<tr>
<td>mayPartition</td>
<td>whether the user can transfer the editing rights for a module to a satellite database (see the chapters on partitions in Using Rational DOORS and Managing Rational DOORS)</td>
</tr>
<tr>
<td>passwordChanged</td>
<td>whether the password has been changed since the account was created</td>
</tr>
<tr>
<td>passwordMayChange</td>
<td>whether the user is permitted to change the password</td>
</tr>
<tr>
<td>mayUseCommandLinePassword</td>
<td>if database restrictions are enabled, whether the user may use the command line password switch</td>
</tr>
<tr>
<td>additionalAuthenticationRequired</td>
<td>whether the user is required to perform additional when logging in (RDS only)</td>
</tr>
</tbody>
</table>
for property in user account

Syntax
for Boolean property Disabled in user do {
    
...
}

where:

<table>
<thead>
<tr>
<th>Boolean property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disabled</td>
<td>whether the user is disabled</td>
</tr>
</tbody>
</table>

Operation
Assigns Boolean property Disabled to each successive user.

isAttribute(user)

Declaration
bool isAttribute(User user, Boolean property Disabled)

Operation
Returns true if the specified user contains the Boolean property Disabled; otherwise, returns false.

isAttribute(user attribute)

Declaration
bool isAttribute(User user, string attribute)
Operation
Returns true if the specified user contains the string \texttt{attribute}; otherwise, returns false.

Example
User \texttt{u} = \texttt{find("Test")}
string \texttt{attr} = "key"
\texttt{bool b = isAttribute(u, attr) }

\texttt{isAttribute(group attribute)}

Declaration
\texttt{bool isAttribute(Group group, string attribute) }

Operation
Returns true if the specified group contains the string \texttt{attribute}; otherwise, returns false.

Example
Group \texttt{g} = \texttt{find("Developers")}
string \texttt{attr} = "key"
\texttt{bool b = isAttribute(g, attr) }

\texttt{delete(user attribute)}

Declaration
\texttt{void delete(User user, string attribute) }

Operation
Deletes the specified string \texttt{attribute} if found within user.

\texttt{delete(group attribute)}

Declaration
\texttt{void delete(Group group, string attribute) }

Operation
Deletes the specified string \texttt{attribute} if found within group.
**delete(user property)**

**Declaration**

```c
void delete(User user, Boolean property Disabled)
```

**Operation**

Deletes the Boolean property Disabled within user. You cannot delete properties of other types. This action takes effect after `saveUserRecord` has been called. It is then permanent and cannot be reversed.

**get(user property)**

**Declaration**

```c
string get(User user, Boolean property Disabled)
```

**Operation**

Returns the value of the Boolean property Disabled within user. If the property does not exist, a DXL run-time error occurs.

**get(user attribute)**

**Declaration**

```c
string get(User user, string attribute)
```

**Operation**

Returns the value of the string attribute within user. If the property does not exist, a DXL run-time error occurs.

**Example**

```c
User u = find(“Test”)
string attr = “key”
string val = get(u, attr)

print val
```

**get(group attribute)**

**Declaration**

```c
string get(Group group, string attribute)
```
Operation

Returns the value of the string attribute within group. If the property does not exist, a DXL run-time error occurs.

Example

Group g = find("Developers")
string attr = "key"
string val = get(g, attr)

print val

set(user property)

Declaration

void set(User user, Boolean property Disabled, string value)

Operation

Updates the value of the Boolean property Disabled within user. If the property does not exist it is created.

set(user attribute)

Declaration

void set(User user, string attribute, string value)

Operation

Updates the value of the string attribute to the specified value. If the attribute does not exist it is created.

Example

User u = find("Test")
string attr = "key"
string val = "value"

set(u, attr, val)

set(group attribute)

Declaration

void set(Group group, string attribute, string value)
Operation
Sets the string attribute to the specified value. If the attribute does not exist it is created.

Example
Group g = find(“Developers”)
string attr = “key”
string val = “value”

set(g, attr, val)

setGroup

Declaration
string setGroup(Group id,
    property,
    {string|bool} value)

Operation
Updates the value of the specified standard property (from the String property table) within the group id.
If successful, returns a null string; otherwise, returns an error message.

setUser

Declaration
string setUser(User user,
    property,
    {string|int|bool} value)

Operation
Updates the value of the specified standard property (from the String property table) within user.
If successful, returns a null string; otherwise, returns an error message.

addGroup

Declaration
string addGroup(string name)

Operation
Creates group name. If the operation is successful, returns a null string; otherwise, returns an error message.
deleteGroup

Declaration

string deleteGroup(Group group)

Operation

Deletes group group from the Rational DOORS database. It does not affect underlying users.

This action takes effect after the user directory has been refreshed using the saveDirectory function. It is then permanent and cannot be reversed.

If the operation is successful, returns a null string; otherwise, returns an error message.

addUser

Declaration

string addUser(string name, string uid)
string addUser(string name, string password, string uid)

Operation

The first form creates a user account with the specified name, and system login, uid. If the operation succeeds returns a null string; otherwise, returns an error message. This function must be used after a call to the createPasswordDialog function, so that the password is set to an initial value. The user must change the password on first use. If there has been no previous call to the createPasswordDialog function, the password is set to a null string.

The second form is only supported for compatibility with earlier releases. It is deprecated because passwords are passed as plain text.

This action takes effect after the user directory has been refreshed using the saveDirectory function.

Example

See the section “Creating a user account example,” on page 228.

deleteUser

Declaration

string deleteUser(User user)

Operation

Deletes the user account for user from the Rational DOORS database. Appropriate e-mails are also issued to the same people who are notified of unsuccessful logins.
This action takes effect after the user directory has been refreshed using the `saveDirectory` function. It is then permanent and cannot be reversed.

If the operation is successful, returns a null string; otherwise, returns an error message.

---

### addMember

**Declaration**

```c
void addMember(Group group, 
               User user)
```

**Operation**

Adds user `user` to group `group`.

This action takes effect after the user directory has been refreshed using the `saveDirectory` function.

---

### deleteMember

**Declaration**

```c
bool deleteMember(Group group, 
                  User user)
```

**Operation**

Deletes user `user` from group `group`. If the operation succeeds, returns `true`; otherwise, returns `false`.

This action takes effect after the user directory has been refreshed using the `saveDirectory` function.

---

### deleteAllMembers

**Declaration**

```c
bool deleteAllMembers(Group group)
```

**Operation**

Deletes all users from group `group`.

This action takes effect after the user directory has been refreshed using the `saveDirectory` function.

---

### member

**Declaration**

```c
bool member(Group group, 
            User user)
```
Operation
If user *user* is a member of group *group*, returns *true*; otherwise returns *false*.

**stringOf(user class)**

**Declaration**
```
string stringOf(UserClass userClass)
```

**Operation**
Returns a string representation of the specified user class. This can be one of the following values:
"Administrator"
"Standard"
"Database Manager"
"Project Manager"
"Custom"

**LDAP**

**saveLdapConfig()**

**Declaration**
```
string saveLdapConfig()
```

**Operation**
Save the LDAP configuration to the database. Returns empty string on success, error message on failure.

**loadLdapConfig()**

**Declaration**
```
string loadLdapConfig()
```

**Operation**
Load the LDAP configuration from the database. Returns empty string on success, error message on failure.
**getUseLdap()**

**Declaration**

```cpp
bool getUseLdap()
```

**Operation**

Gets the value of the flag which determines if we are using LDAP for storage of user and group information.

**setUseLdap()**

**Declaration**

```cpp
string setUseLdap(bool usingLdap)
```

**Operation**

Sets the value of the flag which determines if we are using LDAP for storage of user and group information. Only the administrator can set this value. Returns empty string on success, error message on failure.

**updateUserList()**

**Declaration**

```cpp
string updateUserList()
```

**Operation**

Update the Rational DOORS user list from the LDAP user list. Creates standard users for all the users permitted by LDAP if they do not already exist in the Rational DOORS database, and updates user name and system login name for existing users.

**Note:** This operation can take a long time, particularly if no group of Rational DOORS users has been specified (see `setDoorsUserGroupDN`).

**updateGroupList()**

**Declaration**

```cpp
string updateGroupList()
```

**Operation**

Update the Rational DOORS group list from the LDAP group list. Creates Rational DOORS groups for all the groups permitted by LDAP if they do not already exist in the Rational DOORS database, and updates group name for existing groups.

**Note:** This operation can take a long time, particularly if no group of Rational DOORS groups has been specified (see `setDoorsGroupGroupDN`).
LDAP Configuration

findUserRDNFromName

Declaration

string findUserRDNFromName(string name, bool &unique, string &uid)

Operation

Search for name in the LDAP directory, in the attribute specified by name for Rational DOORS user names, in the Rational DOORS user subtree.

If found, return the distinguished name of the entry, relative to the Rational DOORS user root. Also sets the unique flag true if only one matching entry was found, and fills in the uid string with the system login name obtained from the matching entry. If not found, returns NULL. Only the administrator can run this function.

findUserRDNFromLoginName

Declaration

string findUserRDNFromLoginName(string uid, bool &unique, string &name)

Operation

Search for uid in the LDAP directory, in the attribute specified for system login names, in the Rational DOORS user subtree.

If found, return the distinguished name of the entry, relative to the Rational DOORS user root. Also sets the unique flag true if only one matching entry was found, and fills in the name string with the Rational DOORS user name obtained from the matching entry. If not found, returns NULL. Only the administrator can run this function.

findGroupRDNFromName

Declaration

string findGroupRDNFromName(string name, bool &unique)

Operation

Search for name in the LDAP directory, in the attribute specified for Rational DOORS group names, in the Rational DOORS group subtree.

If found, return the distinguished name of the entry, relative to the Rational DOORS group root. Also sets the unique flag true if only one matching entry was found. If not found, returns NULL. Only the administrator can run this function.
**findUserInfoFromDN**

**Declaration**

```dxl
string findUserInfoFromDN(string dn, string &name, string &uid)
```

**Operation**

Search for an entry with distinguished name `dn` in the LDAP directory.

If found, fills in the name and uid with the Rational DOORS user name and system login name obtained from the matching entry. Returns NULL. Only the administrator can run this function.

**checkConnect**

**Declaration**

```dxl
string checkConnect()
```

**Operation**

Check the current LDAP configuration by attempting to connect to the specified server/port as the user specified by Rational DOORS bind `dn` with the Rational DOORS bind password. Returns NULL on success, error message on failure.

**checkDN**

**Declaration**

```dxl
string checkDN(string dn)
```

**Operation**

Check that the given `dn` is a valid entry in the directory specified by the current LDAP configuration. This can be run to check that the user root, group root, user group `dn`, and group group `dn` have been set to existing values. Only the administrator can run this function.

**Example**

```dxl
LdapItem item
for item in ldapGroupList do
{
    print item.name "\n"
    print item.dn "\n"
    print item.uid "\n"
}

for item in ldapUserList do
```


```dxl
def
    print item.name "\n"
def
    print item.dn "\n"
def
    print item.uid "\n"
}

LDAP server information

getLdapServerName

Declaration
string getLdapServerName()

Operation
Gets the name of the LDAP server.

setLdapServerName(string)

Declaration
string setLdapServerName(string name)

Operation
Sets the name of the LDAP server. Only the administrator can set this value. Returns empty string on success, error message on failure.

getPortNo

Declaration
int getPortNo()

Operation
Gets the port number of the server used for storage of user and group information.

setPortNo

Declaration
string setPortNo(int portNo)
Operation
Sets the port number of the server used for storage of user and group information. Only the administrator can set this value. Returns empty string on success, error message on failure.

getDoorsBindNameDN

Declaration
string getDoorsBindNameDN()

Operation
Gets the dn of the user we use to bind to the LDAP server.

setDoorsBindNameDN

Declaration
string setDoorsBindNameDN(string name)

Operation
Sets the dn of the user we use to bind to the LDAP server. Only the administrator can set this value. Returns empty string on success, error message on failure.

setDoorsBindPassword

Declaration
string setDoorsBindPassword(string pass)

Operation
Sets the password we use to bind to the LDAP server. Only the administrator can set this value. Returns empty string on success, error message on failure.

Note: There is no getDoorsBindPassword as DXL does not need to know this.

setDoorsBindPasswordDB

Declaration
string setDoorsBindPasswordDB(DB parentWindow)

Operation
This presents the user with a password dialog box. If the user enters the same valid password in both fields of the dialog box, the setDoorsBindPassword() functionality is executed.
This returns null on success, and an error string on failure (either if the user does not enter the same valid password in both fields of the dialog box, or if the setting of the password option failed).

### getDoorsUserRoot

**Declaration**

```c
string getDoorsUserRoot()
```

**Operation**

Gets the identifier of the directory subtree used for storage of user information.

### setDoorsUserRoot

**Declaration**

```c
string setDoorsUserRoot(string ident)
```

**Operation**

Sets the identifier of the directory subtree used to search the LDAP server for users. Only the administrator can set this value. Returns empty string on success, error message on failure.

### getDoorsGroupRoot

**Declaration**

```c
string getDoorsGroupRoot()
```

**Operation**

Gets the identifier of the directory subtree used for storage of group information.

### setDoorsGroupRoot

**Declaration**

```c
string setDoorsGroupRoot(string ident)
```

**Operation**

Sets the identifier of the directory subtree used to search the LDAP server for groups. Only the administrator can set this value. Returns empty string on success, error message on failure.
getDoorsUserGroupDN

Declaration
string getDoorsUserGroupDN()

Operation
Gets the dn of the LDAP group used to specify permitted Rational DOORS users.

setDoorsUserGroupDN

Declaration
string setDoorsUserGroupDN(string dn)

Operation
Sets the dn of the LDAP group used to specify permitted Rational DOORS users. Only the administrator can set this value. Returns empty string on success, error message on failure.

getDoorsGroupGroupDN

Declaration
string getDoorsGroupGroupDN()

Operation
Gets the dn of the LDAP group used to specify permitted Rational DOORS groups.

setDoorsGroupGroupDN

Declaration
string setDoorsGroupGroupDN()

Operation
Sets the dn of the LDAP group used to specify permitted Rational DOORS groups. Only the administrator can set this value. Returns empty string on success, error message on failure.
LDAP data configuration

getDoorsUsernameAttribute

Declaration

string getDoorsUsernameAttribute()

Operation

Gets the name of the LDAP attribute to be used for a Rational DOORS user name.

setDoorsUsernameAttribute

Declaration

string setDoorsUsernameAttribute(string name)

Operation

Sets the name of the LDAP attribute to be used for a Rational DOORS user name. Only the administrator can set this value. Returns empty string on success, error message on failure.

getLoginNameAttribute

Declaration

string getLoginNameAttribute()

Operation

Gets the name of the LDAP attribute to be used for the system login name.

setLoginNameAttribute

Declaration

string setLoginNameAttribute(string name)

Operation

Sets the name of the LDAP attribute to be used for the system login name. Only the administrator can set this value. Returns empty string on success, error message on failure.
getEmailAttribute

Declaration
string getEmailAttribute()

Operation
Gets the name of the LDAP attribute to be used for the user’s email address.

setEmailAttribute

Declaration
string setEmailAttribute(string email)

Operation
Sets the name of the LDAP attribute to be used for the user’s email address. Only the administrator can set this value. Returns empty string on success, error message on failure.

descriptionAttribute

Declaration
string getDescriptionAttribute()

Operation
Gets the name of the LDAP attribute to be used for the user’s description.

setDescriptionAttribute

Declaration
string setDescriptionAttribute(string name)

Operation
Sets the name of the LDAP attribute to be used for the user’s description. Only the administrator can set this value. Returns empty string on success, error message on failure.

getTelephoneAttribute

Declaration
string getTelephoneAttribute()
Operation
Gets the name of the LDAP attribute to be used for the user’s telephone number.

setTelephoneAttribute

Declaration
string setTelephoneAttribute(string phone)

Operation
Sets the name of the LDAP attribute to be used for the user’s telephone number. Only the administrator can set this value. Returns empty string on success, error message on failure.

getAddressAttribute

Declaration
string getAddressAttribute()

Operation
Gets the name of the LDAP attribute to be used for the user’s address.

setAddressAttribute

Declaration
string setAddressAttribute(string address)

Operation
Sets the name of the LDAP attribute to be used for the user’s address. Only the administrator can set this value. Returns empty string on success, error message on failure.

getGroupObjectClass

Declaration
string getGroupObjectClass()

Operation
Gets the name of the LDAP object class to be used to identify groups. Typically this value will be groupOfUniqueNames.
setGroupObjectClass

Declaration

string setGroupObjectClass(string class)

Operation

Sets the name of the LDAP object class to be used to identify groups. Only the administrator can set this value. Returns empty string on success, error message on failure.

getGroupMemberAttribute

Declaration

string getGroupMemberAttribute()

Operation

Gets the name of the LDAP attribute to be used to identify group members. Typically this value will be uniqueMember.

setGroupMemberAttribute

Declaration

string setGroupMemberAttribute(string name)

Operation

Sets the name of the LDAP attribute to be used to identify group members. Only the administrator can set this value. Returns empty string on success, error message on failure.

getGroupNameAttribute

Declaration

string getGroupNameAttribute()

Operation

Gets the name of the LDAP attribute to be used for a group’s name. Typically this value will be cn.

setGroupNameAttribute

Declaration

string setGroupNameAttribute(string group)
Operation

Sets the name of the LDAP attribute to be used for a group’s name. Only the administrator can set this value. Returns empty string on success, error message on failure.

Group and user properties

Declaration

string ldapRDN

If we have a user u, print u.ldapRDN prints the user’s LDAP relative distinguished name, which may be empty if LDAP is not being used.

The administrator can set a user’s LDAP rdn with

u.ldapRDN = new value.

string utf8(ansiString)

Declaration

string utf8(string ansiString)

Operation

This returns the UTF-8 format conversion of an ANSI string argument ansiString. LDAP servers use UTF-8 encoding, whereas Rational DOORS data is stored in ANSI format. This affects the encoding of extended characters, such as accented letters, which are encoded in UTF-8 as 2-byte sequences.

string ansi(utf8String)

Declaration

string ansi(string utf8String)

Operation

This returns the ANSI format conversion of a UTF-8 string argument utf8String. LDAP servers use UTF-8 encoding, whereas Rational DOORS data is stored in ANSI format. This affects the encoding of extended characters, such as accented letters, which are encoded in UTF-8 as 2-byte sequences.

Rational Directory Server

After using any of the following functions to modify the Rational Directory Server, use the saveLdapConfig() function to save the modifications.
**getUseTelelogicDirectory**

**Declaration**

```dxl
bool getUseTelelogicDirectory()
```

**Operation**

Returns a flag indicating whether Rational Directory Server support is enabled.

---

**setUseTelelogicDirectory**

**Declaration**

```dxl
string setUseTelelogicDirectory(bool b)
```

**Operation**

Enables or disables Rational Directory Server support.

- Returns an error string if the current user is not the administrator.
- Returns an error message if the argument is `true` and ordinary LDAP is already enabled.

---

**getTDServerName**

**Declaration**

```dxl
string getTDServerName()
```

**Operation**

Returns the Rational Directory Server name.

---

**setTDServerName**

**Declaration**

```dxl
string setTDServerName(string s)
```

**Operation**

Sets the Rational Directory Server name.

- Returns an error string if the current user is not the administrator.
getTDPortNo

Declaration
int getTDPortNo()

Operation
Returns the Rational Directory Server port number.

setTDPortNo

Declaration
string setTDPortNo(int i)

Operation
Sets the Rational Directory Server port number.
Returns an error string if the current user is not the administrator.

getTDBindName

Declaration
string getTDBindName()

Operation
Returns the Rational Directory Server administrator bind (login) name.

setTDBindName

Declaration
string setTDBindName(string s)

Operation
Sets the Rational Directory Server administrator bind (login) name.
Returns an error string if the current user is not the administrator.

setTDBindPassword

Declaration
string setTDBindPassword(string s)
Operation
Sets the Rational Directory Server administrator bind (login) password.
Returns an error string if the current user is not the administrator.

setTDBindPassword

Declaration
string setTDBindPassword(DB bind_pass)

Operation
Sets the Rational Directory Server administrator bind (login) password from the specified database.

getTDUseDirectoryPasswordPolicy

Declaration
bool getTDUseDirectoryPasswordPolicy()

Operation
Returns a flag indicating whether the directory should handle all password policy issues.

setTDUseDirectoryPasswordPolicy

Declaration
string setTDUseDirectoryPasswordPolicy(bool TD_dir)

Operation
Enables or disables support for the directory password policy.
Returns an error string if the current user is not the administrator.

getAdditionalAuthenticationEnabled

Declaration
bool getAdditionalAuthenticationEnabled()

Operation
Returns true if enhanced security users need to perform additional authentication during login. Only relevant when authentication is being controlled via RDS.
getAdditionalAuthenticationPrompt

Declaration

string getAdditionalAuthenticationPrompt()

Operation

Returns the label under which additional authentication is requested, if enhanced security is enabled, for example the label for the second “password” field. Only relevant when authentication is being controlled via RDS.

getSystemLoginConformityRequired

Declaration

bool getSystemLoginConformityRequired()

Operation

Returns true if enhanced security users have their system login verified when logging in. Only relevant when authentication is being controlled via RDS.

getCommandLinePasswordDisabled

Declaration

bool getCommandLinePasswordDisabled()

Operation

Return true if the -P command line password argument is disabled by default.

setCommandLinePasswordDisabled

Declaration

string getCommandLinePasswordDisabled(bool)

Operation

Sets whether the -P command line password argument is disabled by default. Supplying true disables the option by default.
Chapter 15
Rational DOORS hierarchy

This chapter describes features that are relevant to items, folders, and projects within the Rational DOORS hierarchy. Features specific to modules and objects are described in the following chapters:

- About the Rational DOORS hierarchy
- Item access controls
- Hierarchy clipboard
- Hierarchy information
- Hierarchy manipulation
- Items
- Folders
- Projects
- Looping within projects

About the Rational DOORS hierarchy

Within a Rational DOORS database there are items, which can be folders, projects, and modules. A project is a special type of folder. The database root is also a folder.

In DXL, the Rational DOORS hierarchy is represented by the data types Item, Folder, Project, and a call to the module function. Open modules are also represented by the Module data type.

Functions that operate on items have equivalents for folders, projects and modules.

Modules and folders are in general referenced by their unqualified names (without paths). However, DXL scripts can specify fully qualified names, which are distinguished by the inclusion of one or more slash (/) characters. These names can be either relative to the current folder, for example:

```
../folder/module
```

or absolute (with a leading slash), for example:

```
/folder/module
```

Create functions fail if an invalid (non-existent) path is specified.

Functions common to all hierarchy items are described in “Hierarchy clipboard,” on page 267, “Hierarchy information,” on page 270, and “Hierarchy manipulation,” on page 274.

Functions specific to items of type Item are described in “Items,” on page 276.

Functions specific to folders are described in “Folders,” on page 279.

Functions specific to projects are described in “Projects,” on page 282.
Item access controls

This section describes functions that report on access rights for items.

canCreate(item)

Declaration

bool canCreate({Item i|Folder f})

Operation

Returns true if the current Rational DOORS user has create access to the item or folder specified by the argument. Otherwise, returns false.

canControl(item)

Declaration

bool canControl({Item i|Folder f})

Operation

Returns true if the current Rational DOORS user can change the access controls on the item or folder specified by the argument. Otherwise, returns false.

canRead(item)

Declaration

bool canRead({Item i|Folder f})

Operation

Returns true if the current Rational DOORS user can read the item or folder specified by the argument. Otherwise, returns false.

canModify(item)

Declaration

bool canModify({Item i|Folder f})
Operation

Returns true if the current Rational DOORS user can modify the item or folder specified by the argument. Otherwise, returns false.

canDelete(item)

Declaration

bool canDelete(Item i|Folder f)

Operation

Returns true if the current Rational DOORS user can delete the item or folder specified by the argument. Otherwise, returns false.

Hierarchy clipboard

This section defines functions for the hierarchy clipboard. Passing a null argument of type Item, Folder, or Project to any function, or a null string to a call to the module function results in a run-time DXL error. The term item means a variable of type Item, type Folder, or type Project, or a call to the module function.

clipCut

Declaration

string clipCut(Item i)

Operation

Places a write lock on the item specified by the argument, and adds it to the clipboard as part of a set of cut items. If the write lock fails, or if the user does not have delete access to the item and its descendants (if any), the call to clipCut fails.

If the previous operation was not a cut, this function first clears the clipboard. If the item is deleted, returns an error message.

No other user can open the cut item until it has been pasted or the cut has been undone.

clipCopy

Declaration

string clipCopy(Item i)
Operation
Places a share lock on the item specified by the argument, and adds it to the clipboard as part of a set of copied items. If the share lock fails, or if the user does not have read access to the item, the call to clipCopy fails. Any descendants of the item to which the user does not have read access are not included as part of the set of items placed on the clipboard.

If the previous operation was a paste, this function first clears the clipboard. If the previous operation was a cut, this function first performs an undo. If the item is deleted, returns an error message.

No other user can move, delete or rename the item until it has been pasted or the copy has been undone.

clipClear

Declaration
string clipClear([bool force])

Operation
If the last operation was not a cut, unlocks and clears the clipboard contents. If the last operation was a cut, the result depends on the value of force as follows:

- false: the call fails
- true: purges the contents of the clipboard from the database.

If you omit force, its value is assumed to be false.

clipPaste

Declaration
string clipPaste(Folder folderRef)

Operation
Pastes the contents of the clipboard to folderRef. If the user does not have create access to the destination, the call to clipPaste fails. If folderRef is deleted, returns an error message.

If the previous operation was a cut, moves the contents of the clipboard from their original location, and places a share lock on them. Otherwise, unlocks the originals, and makes copies of them in folderRef. In this case, any projects have Copy of in front of their names, because duplicate project names are not allowed. If this still results in duplicate names, Copy n of is used, where n is the lowest number >= 2 that prevents duplication. This function uses the same naming convention to avoid duplication when copying items into their original folder.

The items pasted from the clipboard remain share locked until the clipboard is cleared. This is done automatically when the client closes down, or when the user opens any module in the clipboard for exclusive edit, or deletes, renames, or moves any item in the clipboard.
clipUndo

**Declaration**

string clipUndo({Item i})

**Operation**

If the last operation was a cut or copy, unlocks and clears the clipboard contents.

clipLastOp

**Declaration**

int clipLastOp()

**Operation**

Returns an integer indicating the last operation performed on the hierarchy clipboard. The returned value can be of: Cut, Copy, Clear, Paste, Undo.

itemClipboardIsEmpty

**Declaration**

bool itemClipboardIsEmpty()

**Operation**

If there are no items in the hierarchy clipboard, returns true; otherwise, returns false.

inClipboard

**Declaration**

bool inClipboard({Item i|Folder f|Project p|Module m|ModName_ modRef})

**Operation**

If the item specified by the argument is in the hierarchy clipboard, returns true; otherwise, returns false.
Hierarchy information

This section defines functions that provide information about items, folders, projects, or modules. The term *item* means a variable of type Item, type Folder, type Project or type ModName_. You can also reference an open module using the data type Module. Passing a null argument of type Item, Folder, Project, Module or ModName_ to any function results in a run-time DXL error.

folder, project, module(state)

**Declaration**

```dxl
bool folder(string folderName)
bool project(string projectName)
bool module(string moduleName)
```

**Operation**

Returns true if the argument is the name of a folder, project, or module to which the current user has read access; otherwise, returns false.

Because a project is a special class of folder, the `folder` function returns true for projects as well as other folders.

description

**Declaration**

```dxl
string description({Item i|Folder f|Project p|Module m|ModName_ modRef})
```

**Operation**

Returns the description of the item specified by the argument.

**Example**

```dxl
print description current Module
```

name(item)

**Declaration**

```dxl
string name({Item i|Folder f|Project p|Module m|ModName_ modRef})
```

**Operation**

Returns the unqualified name of the item specified by the argument.

**Example**

```dxl
print name current Module
```
**fullName(item)**

**Declaration**

```dxl
string fullName({Item i|Folder f|Project p|Module m|ModName_ modRef})
```

**Operation**

Returns the full name of the item specified by the argument, including the path from the nearest ancestor project, or if not inside a project, from the root folder.

---

**path(item)**

**Declaration**

```dxl
string path({Item i|Folder f|Project p|Module m|ModName_ modRef})
```

**Operation**

Returns the full name of the parent of the item specified by the argument from the nearest ancestor project, or if not inside a project, from the root folder.

---

**getParentFolder(item)**

**Declaration**

```dxl
Folder getParentFolder({Item i|Folder f|Project p|Module m|ModName_ modRef})
```

**Operation**

Returns the folder containing the item specified by the argument. If the argument is the root folder, returns null.

---

**getParentProject(item)**

**Declaration**

```dxl
Project getParentProject({Item i|Folder f|Project p|Module m|ModName_ modRef})
```

**Operation**

Returns the nearest ancestor project for the item specified by the argument, or null if there is none. If the item is a project, this function does not return the project itself, but the nearest one above (or null if there is none).

---

**isDeleted(item)**

**Declaration**

```dxl
bool isDeleted({Item i|Folder f|Project p|ModName_ modRef})
```
Operation
If the item specified by the argument is marked as deleted or soft deleted, or if it does not exist, or if the user does not have read access to it, returns true; otherwise, returns false.

```
setShowDeletedItems(bool)
```

Declaration
void setShowDeletedItems(bool show)

Operation
If bool show is set to true, deleted items will be visible in the Database Explorer. Setting show to false hides all deleted items.

```
type
```

Declaration
string type({Item i|Folder f|Module m|ModName modRef})

Operation
Returns the type of the item specified by the argument as a string. Possible values are shown in the following table.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Item</th>
<th>Folder</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Folder&quot;</td>
<td>y</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>&quot;Project&quot;</td>
<td>y</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>&quot;Formal&quot;</td>
<td>y</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>&quot;Link&quot;</td>
<td>y</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>&quot;Descriptive&quot;</td>
<td>y</td>
<td>n</td>
<td>y</td>
</tr>
</tbody>
</table>

Example
print type(item "/")

```
uniqueID
```

Declaration
string uniqueID({Item i|Folder f|Project p|ModName modRef|Module m})
**Operation**

Returns a unique identifier for the specified item, which lasts for the lifetime of the item, and is never reused. The unique identifier does not change when the item is moved or renamed. If the item is copied, the copy has a different identifier. A call to this function where \(i\) does not exist causes a DXL run-time error.

**qualifiedUniqueID**

**Declaration**

```dxl
string qualifiedUniqueID({Item i|Folder f|Project p|ModName_ name|Module m})
```

**Operation**

Returns a representation of a reference to the specified Item, Folder, Project, Module or ModName_, which uniquely identifies that object amongst databases.

Provided that supported mechanisms for the creation of Rational DOORS databases are used, these unique identifiers can be treated as globally unique; no two objects in any two databases will have the same qualifiedUniqueID.

See also uniqueID, which returns an unqualified representation of a reference.

**getReference**

**Declaration**

```dxl
string getReference(Item referrer, Item referee)
```

**Operation**

Returns a reference to the referee from the referrer. This reference is invariant under archive/restore (both inter-database and intra-database) and copy/paste. Such a reference is to be used in preference to the referee’s index, unless the reference is intended to be variant under such operations.

**itemFromReference**

**Declaration**

```dxl
Item itemFromReference(Item referrer, string ref)
```

**Operation**

Returns the item to which \(ref\) refers from the specified referrer. \(ref\) must be a string that was obtained using the getReference() perm. If the reference cannot be resolved, the returned item will satisfy null.

**Example**

Make a reference from the current module to an item named “a”

```dxl
Item i = item fullName current Module
Item j = item "a"
```
// rj is a reference to j from i
string rj = getReference(i, j)

print rj "\n"

This reference will never change when i and j are moved, copied (together), archived, and restored (together).

Copy i and j to get ii and jj

Item j = itemFromReference(i, rj) // get item that rj refers
Item jj = itemFromReference(ii, rj) // get item that rj refers

Typically these would be used when generating traceability. The DXL that generates the layout DXL or attribute DXL would call getReference and then insert the returned value into the layout DXL or attribute DXL code as the value passed to itemFromReference().

## Hierarchy manipulation

This section defines functions for item manipulation. All creation functions are specific to the type of item being created, but you can delete, undelete, purge, move, and rename items of all types using the Item handle. The term item means a variable of type Item, type Folder, type Project or type ModName_. You can also reference an open module using the data type Module. Passing a null argument of type Item, Folder, Project, Module or ModName_ to any function results in a run-time DXL error.

### delete(item)

**Declaration**

```dxl
string delete({Item i|Folder f|Project p})
string delete(ModName_ &modRef
    [,bool hardDelete])
bool delete(ModName_ &modRef)
```

**Operation**

Marks the item specified by the argument as deleted. If the item is already marked as deleted, or if the user does not have delete access to it, the call fails.

The first and second forms return a null string on success; otherwise, an error message.

In the second form, if hardDelete is set to false, the module is not purged. If hardDelete is true or missing and if the module was soft-deleted, the module is purged. If the module was not soft-deleted, the function returns an error message. If the operation succeeds and the module is purged, the function also sets the ModName_ argument to null.
The third form is retained for compatibility with earlier releases. It returns `true` on success; otherwise, `false`. This is equivalent to `hardDelete(module)` (the module need not be soft deleted). If the operation succeeds, also sets the `ModName_` argument to `null`.

For a folder or project, the user must also have delete access to all the undeleted folders, projects, and modules in it.

### undelete(item)

**Declaration**

```dxl
string undelete({Item i|Folder f|Project p|ModName_ modRef})
bool undelete(ModName_ modRef)
```

**Operation**

Marks the item specified by the argument as undeleted. If the item is not marked as deleted, or if the user does not have delete access to the item, the call fails.

The first form returns a null string on success; otherwise, an error message.

The second form is retained for compatibility with earlier releases. It returns `true` on success; otherwise, `false`.

For a folder or project, this function also marks as undeleted all folders, projects, and modules in it, to which the user has delete access.

**Example**

```
undelete item "my folder"
```

### purge(item)

**Declaration**

```dxl
string purge({Item &i|Folder &f|Project &p|ModName_ &modRef})
bool purge(ModName_ &modRef)
```

**Operation**

Purges the item specified by the argument from the database. If the operation succeeds, sets the argument to `null`. If the item is not marked as deleted, or if the user does not have delete access to the item, the call fails.

The first form returns a null string on success; otherwise, an error message.

The second form is retained for compatibility with earlier releases. It returns `true` on success; otherwise, `false`.

For a folder or project, the user must also have delete access to all the undeleted folders, projects, and modules in it.

For a `ModName_` argument, the function deletes all incoming and outgoing links before purging the module.

**Example**

```
purge item "my folder"
```

or
Item i = item "my folder"
purge i

move(item)

Declaration
string move({Item i|Folder f|Project p|ModName_ modRef}, Folder destination)

Operation
Moves the item specified by the first argument to folder destination. The folder can be any folder except the database root.
If the user does not have delete access to the item, or create access to the destination folder, the call fails.
If the operation succeeds, returns a null string; otherwise, returns a string describing the error.

Example
move(item "My Module", folder "/new projects")

rename(item)

Declaration
string rename({Item i|Folder f|Project p|ModName_ modRef}, string name, string description)
bool rename(ModName_ modRef)

Operation
 Renames the item specified by the first argument to name and associates it with description. The name argument must be an unqualified name. If the user does not have modify access to the item, the call fails.
The first form returns a null string on success; otherwise, an error message.
The second form is retained for compatibility with earlier releases. It returns true on success; otherwise, false.

Example
rename(folder "my folder", "public", "for review")

Items

This section defines functions and for loops for items, which make use of the Item data type. Passing a null argument of type Item to any function results in a run-time DXL error.
See also the functions in “Hierarchy clipboard,” on page 267, “Hierarchy information,” on page 270, and “Hierarchy manipulation,” on page 274.

item(handle)

**Declaration**

Item item(string itemName)

**Operation**

If itemName is the name of an item to which the current user has read access, returns a handle of type Item; otherwise, returns null.

itemFromID(handle)

**Declaration**

Item itemFromID(string uniqueID)

**Operation**

If uniqueID is the ID of an item to which the current user has read access, returns a handle of type Item; otherwise, returns null.

for item in folder

**Syntax**

for itemRef in folder do {
  ...
}

where:

itemRef is a variable of type Item

folder is a variable of type Folder

**Operation**

Assigns itemRef to be each successive undeleted item (for which the user has read access) in folder. Items in sub-folders are not included.

**Example**

Item i

for i in current Folder do {
  print (name i) "\n"
}
for all items in folder

Syntax

```dxl
for itemRef in all folder do {
  ...
}
```

where:

- `itemRef` is a variable of type `Item`
- `folder` is a variable of type `Folder`

Operation

Assigns `itemRef` to be each successive item (for which the user has read access) in `folder`, including deleted items. Items in sub-folders are not included.

Example

```dxl
Folder f = current
Item itemRef

for itemRef in f do {
  print fullName(itemRef) "\n"
}
```

for all items in project

Syntax

```dxl
for itemRef in project do {
  ...
}
```

where:

- `itemRef` is a variable of type `Item`
- `project` is a variable of type `Project`

Operation

Assigns `itemRef` to be each successive undeleted item (for which the user has read access) in `project`, looping recursively through contained folders and projects.
Example

```dxl
Item itemRef

for itemRef in current Project do
    print name(itemRef) "\n"
```

Folders

This section defines functions for folders.

See also the functions in “Hierarchy clipboard,” on page 267, “Hierarchy information,” on page 270, and “Hierarchy manipulation,” on page 274.

Setting current folder

The assignment operator = can be used as shown in the following syntax:

```dxl
current = Folder folder
```

Makes `folder` the current folder, provided the user has read access to the folder. See also, the `current(folder)` function.

To set the current folder to the database root, use:

```dxl
current = folder "/"
```

For large DXL programs, when you set the current folder, cast the current on the left hand side of the assignment to the correct type. This speeds up the parsing of the DXL program, so when your program is first run, it is loaded into memory quicker. It does not affect the subsequent execution of your program. So:

```dxl
current = newCurrentFolder
```

becomes

```dxl
(current FolderRef__) = newCurrentFolder
```

Note that this cast only works for assignments to current. It is not useful for comparisons or getting the value of the current folder.

`current(folder)`

**Declaration**

```dxl
Folder current()
```

**Operation**

Returns a handle on the current folder.

The current folder can be a project.

The current folder has two important implications:
• When you specify an item name, it is interpreted relative to the current folder.
• When you set the current folder using the assignment operator, you lock that folder and its ancestors, so that it cannot be renamed, deleted or moved.

The project or folder that is opened in the Database Explorer is similarly locked. If you open a DXL window or run another DXL script, that has its own current folder. The current folder for the DXL window is initially the current folder of its parent.

If all folders are closed, the database root becomes the current folder.

**Example**

Folder f = current

---

**folder(handle)**

**Declaration**

Folder folder(string folderName)
Folder folder(Item itemRef)

**Operation**

If the argument specifies a folder to which the current user has read access, returns a handle of type Folder; otherwise, returns null.

The string "/" identifies the database root.

**Example**

This example sets the current folder to the database root:

current = folder "/"

---

**convertProjectToFolder**

**Declaration**

string convertProjectToFolder(Project projectRef, Folder &folderRef)

**Operation**

Converts the project projectRef to a folder folderRef. If the operation succeeds, sets projectRef to null, makes the folder argument valid, and returns a null string; otherwise, returns an error message. If the user does not have control access to the project or the create projects power (through mayCreateTopLevelFolders), the call fails.

**Example**

Project p = project "/Construction Project"
Folder f
string s = convertProjectToFolder(p, f)
if (null s)
    print "Converted project ", name(f), "to folder."
else
    print "Error: ", s

cvtColorFolderToProject

Declaration

string
cvtColorFolderToProject(Folder folderRef,
            Project &projectRef)

Operation

Converts the folder folderRef to a project projectRef. If the operation succeeds, sets folderRef to null, makes the project argument valid, and returns a null string; otherwise, returns an error message. If the user does not have control access to the folder or the create projects power (through mayCreateTopLevelFolders), the call fails.

Example

Folder f = folder "/Construction Project/test records"
Project p
string s = cvtLotFolderToProject(f, p)
if (null s)
    print "Converted folder ", name(p), "to project."
else
    print "Error: ", s

create(folder)

Declaration

Folder create(string name,
            string description)

string create(string name, description desc, Folder& f)

Operation

Creates a folder with the given name and description. The name argument can be an absolute or relative name, and may include the path. If the user does not have create access to the parent folder, the call fails.

The second form of the perm performs the same function as the first, but returns any error message, and passes the created folder back via the last argument.
closeFolder

**Declaration**

```dxl
closeFolder() : string
```

**Operation**

Changes the current folder to refer to the parent of the current folder. If the operation succeeds returns a null string; otherwise, returns a string describing the error.

**Example**

```dxl
closeFolder()
```

---

**Projects**

This section defines operators, functions and for loops for projects, which make use of the `Project` data type. Passing a `null` argument of type `Project` to any function results in a run-time DXL error.

See also the functions in “Hierarchy clipboard,” on page 267, “Hierarchy information,” on page 270, and “Hierarchy manipulation,” on page 274.

---

**Setting current project**

The assignment operator `=` can be used as shown in the following syntax:

```dxl
(current ) = Project project
```

Makes `project` the current folder, and the current project, provided the user has read access to the folder. See also, the `current(project)` function.

If the current folder is a project, it is also the current project. If the current folder is not a project, the current project is the nearest project containing the current folder. If the current folder is not contained in a project, the current project is `null`.

For large DXL programs, when you set the current project, cast the current on the left hand side of the assignment to the correct type. This speeds up the parsing of the DXL program, so when your program is first run, it is loaded into memory quicker. It does not affect the subsequent execution of your program. So:

```dxl
(current ) = newCurrentProject
```

becomes

```dxl
(current FolderRef__) = newCurrentProject
```

Note that this cast only works for assignments to current. It is not useful for comparisons or getting the value of the current project.

**Example**

```dxl
(current) = project "/My Project"
```
current(project)

**Declaration**

Project current()

**Operation**

Returns a handle on the nearest ancestor project of the current folder, or null if the current folder is not in any project.

**Example**

Module m

// check project is open
if (null current Project) {
    ack "No project is open"
    halt
}

for m in current Project do {
    print "Module " m."Name" " is open"
}

project(handle)

**Declaration**

Project project(string projectName)

Project project(Item itemRef)

**Operation**

If the argument specifies a project to which the current user has read access, returns a handle of type Project to the project; otherwise, returns null.

for project in database

**Syntax**

for project in database do {
    ...
}

where:

*project* is a variable of type *Project*
Operation

Assigns \textit{project} to be each successive project (for which the user has read access) in the database, excluding deleted projects. Compare with \texttt{for all projects in database}.

Example

This example prints a list of projects in the database:

\begin{verbatim}
Project p
for p in database do {
    print(name p) "\n"
}
\end{verbatim}

\texttt{for all projects in database}

Syntax

\begin{verbatim}
for name in database do {
    ...
}
\end{verbatim}

where:

\begin{itemize}
  \item \textit{name} is a string variable
\end{itemize}

Operation

Assigns the string \textit{name} to be each successive project name (for which the user has read access) in the database, including deleted projects. Compare with \texttt{for project in database}.

Example

This example prints a list of projects in the database:

\begin{verbatim}
string s
for s in database do {
    print s "\n"
}
\end{verbatim}

getInvalidCharInProjectName

Declaration

\begin{verbatim}
char getInvalidCharInProjectName(string s)
\end{verbatim}

Operation

Returns any character in string \textit{s} that would be invalid in a project name.
isDeleted(project name)

**Declaration**

```dxl
bool isDeleted(string projectName)
```

**Operation**

If `projectName` is a project that has been deleted but not purged, or if it does not exist, or if the user does not have read access to it, returns `true`; otherwise, returns `false`.

This function is retained only for compatibility with earlier releases. New programs should use the `isDeleted(item)` function.

**Example**

```dxl
Project p = project "Test Project"
if (!null p && !isDeleted p)
    current = p
```

isValidName

See “isValidName,” on page 296.

create(Project)

**Declaration**

```dxl
Project create(string projName,
               string description
               [,string adminUser,
               [,string password,
               string loginsystem,
               int passwordPolicy,
               int adminPolicy,
               string &message]])
```

```dxl
string create(string name, description desc, Project& p)
```

**Operation**

Creates a project, `projName`, having `description`. The `adminUser` and following arguments are retained for compatibility with earlier releases; in Rational DOORS 6.0, the values of these arguments are ignored. However, a call to `create` that uses any of the legacy arguments sets the current folder to the new project (for compatibility with legacy DXL scripts, which expect the new project to be opened).

You must assign this function to a variable of type `Project`, otherwise, it tries to create a linkset between modules `projName` and `description`.

Administrator power is required for this function.
The second form of the perm performs the same function as the original perm, but returns any error message, and passes the created project back via the last argument.

**Example**

```dxl
create("Test Project", "Play area for DOORS")
```

---

### closeProject

**Declaration**

```dxl
void closeProject()
```

**Operation**

Sets the parent of the current project to be the new current folder. In Rational DOORS 6.0, closing a project means changing the current folder.

**Example**

```dxl
closeProject()
```

---

### openProject

**Declaration**

```dxl
string openProject(string projName [,string user, string pass])
```

**Operation**

Sets the named project as the current folder. The `user` and `password` arguments are retained for compatibility with earlier releases. In Rational DOORS 6.0 these arguments are ignored.

If the project opens successfully, returns `null`; otherwise returns an error message. If the project does not exist, or the user does not have read access to it, the call fails.

**Example**

```dxl
string mess = openProject("Demo", "Catrina Magali", "aneb1r")
```

---

### doorsVersion

**Declaration**

```dxl
string doorsVersion()
```

**Operation**

Returns the version of the current Rational DOORS executable as a string.
Example

print doorsVersion

Looping within projects

The following sections describe the for loops available for looping within projects:

- for all items in project
- for open module in project
- for all modules in project
- for in-partition in project
- for out-partition in project
- for partition definition in project
- for trigger in project
Chapter 16

Modules

This chapter describes features that operate on Rational DOORS modules:

- Module access controls
- Module references
- Module information
- Module manipulation
- Module display state
- Baselines
- Baseline Set Definition
- Baseline Sets
- History
- Descriptive modules
- Recently opened modules
- Module Properties

Module access controls

This section describes functions that report on access rights for a module. The module has to be open in exclusive edit mode.

**canCreate(module)**

**Declaration**

```dxl
bool canCreate(Module m)
```

**Operation**

Returns `true` if the current Rational DOORS user has create access to module `m`; otherwise, returns `false`.

**canControl(module)**

**Declaration**

```dxl
bool canControl(Module m)
```
Operation
Returns true if the current Rational DOORS user can change the access controls on module \( m \); otherwise, returns false.

\[ \text{canModify(module)} \]

Declaration

```
bool canModify(Module m)
```

Operation

Returns true if the current Rational DOORS user can modify module \( m \); otherwise, returns false.

\[ \text{canDelete(module)} \]

Declaration

```
bool canDelete(Module m)
```

Operation

Returns true if the current Rational DOORS user can delete module \( m \); otherwise, returns false.

---

**Module references**

This section defines functions and for loops that make use of the Module data type.

See also the functions in “Hierarchy clipboard,” on page 267.

**Setting current module**

The assignment operator = can be used as shown in the following syntax:

```
current = Module module
```

Makes \( module \) the current module. See also, the current (module) function.

For large DXL programs, when you set the current module, cast the current on the left hand side of the assignment to the correct type. This speeds up the parsing of the DXL program, so when your program is first run, it is loaded into memory quicker. It does not affect the subsequent execution of your program. So:

```
current = newCurrentModule
```

becomes

```
(current ModuleRef__) = newCurrentModule
```
Note that this cast only works for assignments to current. It is not useful for comparisons or getting the value of the current module.

**current(module)**

**Declaration**

```
Module current()
```

**Operation**

Returns a reference to the current module. In some contexts `current` could be ambiguous, in which case it should be followed by `Module` in a cast.

**Example**

```
print (current Module)."Description" "\n"
```

**module(handle)**

**Declaration**

```
Module module(Item itemRef)
ModName_ module(string modRef)
```

**Operation**

The first form returns a handle of type `Module` for `itemRef` if `itemRef` is an open module. Otherwise, it returns `null`.

The second form returns a handle of type `ModName_` for the named module, whether it is open or closed.

**for module in database**

**Syntax**

```
for m in database do {
  ...
}
```

**Operation**

Assigns the variable `m` to be each successive open module (for which the user has read access) in the database.
for open module in project

Syntax

for m in project do {
  ...
}

where:

m is a variable of type Module
project is a variable of type Project

Operation

Assigns the variable m to be each successive open module (for which the user has read access) in project. This loop includes modules in sub folders as well as those in the top level of the project. It does not include modules in projects that are contained in the project. This only works on the user’s computer.

Example

Module m
int count = 0
for m in current Project do {
  print m."Name" "\n"
  count++
}
if (count==0)
  print "no modules in current project\n"

for all modules in project

Syntax

for moduleName in project do {
  ...
}

where:

moduleName is a string variable
project is a variable of type Project
Operation

Assigns the variable `moduleName` to be each successive module name (for which the user has read access) in `project`. This loop includes open or closed modules but only at the top level of the project. This is no longer everything contained in the project. This only works on the user’s computer.

Example

```dxl
string modName
for modName in current Project do
    print modName "\n"
```

for Module in Folder do

Syntax

```dxl
for m in folder do {
    ...
}
```

where:

- `m` is a variable of type `Module`
- `folder` is a variable of type `Folder`

Operation

This provides access to all open modules that have the specified folder as their parent.

Example

```dxl
Module m
Folder f = current
for m in f do {
    print "Module " (name m) " is open "\n"
}
```

Module information

This section defines functions that return information about Rational DOORS modules.

See also the functions in “Hierarchy information,” on page 270.
Module state

Declaration

bool baseline(Module m)
bool exists(ModName_ modRef)
bool open(ModName_ modRef)
bool unsaved(Module m)

Operation

Each function returns true for a condition defined by the function name as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns true if</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>module $m$ is a baseline; otherwise, returns false</td>
</tr>
<tr>
<td>exists</td>
<td>module $modRef$ exists in the current project; otherwise, returns false</td>
</tr>
<tr>
<td>open</td>
<td>module $modRef$ is open in any mode; otherwise, returns false</td>
</tr>
<tr>
<td>unsaved</td>
<td>module $m$ has not been saved since changes were made; otherwise returns false</td>
</tr>
</tbody>
</table>

Example

string s = "/proj1/SRD"
Item i = item s
if (exists module s) print "and the system requirements ... \n"
if (open module s) print "SRD is open\n"

version

Declaration

string version(Module m)

Operation

Returns the version of open module $m$ as a string.

Example

print (version current Module)
canRead, canWrite(module)

Declaration

```dxl
bool canRead(Module m)
bool canWrite(Module m)
```

Operation

Returns whether the current Rational DOORS user has read or write access to the top of open module m.

gerSelectedCol

Declaration

```dxl
int getSelectedCol(Module m)
```

Operation

Returns the integer identifier for the currently selected column in m. If the specified module is not displayed, or no column is selected, returns -1.

isRead, isEdit, isShare

Declaration

```dxl
bool isRead(Module m)
bool isEdit(Module m)
bool isShare(Module m)
```

Operation

Returns whether module m is open for reading, for editing or in shared mode. Otherwise, returns false.

These functions only return values for modules opened by the current user in the current session.

Example

```dxl
Module m
for m in current Project do {
    if (isEdit m)
        print m."Name" " is open edit\n"
}
getInvalidCharInModuleName

Declaration

char getInvalidCharInModuleName(string s)

Operation

Returns any character in string s that would be invalid in a module name.

isValidDescription

Declaration

bool isValidDescription(string descString)

Operation

Returns true if descString is a legal description for a project, module, view or page layout; otherwise, returns false.

Example

This example returns true.

bool b = isValidDescription("Test Description")

isValidName

Declaration

{char|bool} isValidName(string nameString)

Operation

By default, returns the first illegal character of nameString. If you force a type bool, returns true if nameString is a legal name for a project, module, view or page layout; otherwise, returns false.

Example

This example returns &, the first illegal character in the name:

char c = isValidName("illegal&Name")

This example returns true:

char c = isValidName("legalName")

isValidPrefix

Declaration

bool isValidPrefix(string prefixString)
Operation

Returns true if `prefixString` is a legal prefix for an object; otherwise returns false.

Example

This example returns true:

```dxl
bool b = isValidPrefix("PREFIX-1")
```

isVisible

Declaration

```dxl
bool isVisible(Module m)
```

Operation

Returns true if module `m` is open for display on the screen. Otherwise, returns false.

Module manipulation

This section defines the functions for creating modules and performing database administration tasks on modules other than descriptive modules, which are covered in “Descriptive modules,” on page 341.

See also the functions in “Hierarchy manipulation,” on page 274.

create(formal module)

Declaration

```dxl
Module create(string name,
    string desc,
    string prefix,
    int absno
    [,bool display])
```

```dxl
string create(string name, description desc, prefix pref, int absnum, Module& m)
```

Operation

Creates a formal module with name `name`, description `desc`, object prefix `prefix` and starting absolute number `absno`. The `name` argument can be an absolute or relative path. The optional last argument controls whether the module is displayed in the user interface after it has been created.

The second form creates a formal module. However, in the case of an error which causes no module to be created, the error message is returned instead of generating a run-time DXL error.
create(descriptive module)

Declaration

string create(string name, description desc, prefix pref, int absnum, string filename, Module& m)

Operation

Creates a Descriptive module. When an error occurs, which causes no module to be created, the error message is returned instead of generating a run-time DXL error.

create(link module)

Declaration

Module create(string name, string desc, int mapping [,bool display])
string create(string name, description desc, int mapping, Module& m)
const int manyToMany
const int manyToOne
const int oneToMany
const int oneToOne

Operation

Creates a link module with name name, description desc, and a mapping. The name argument can be an absolute or relative path. The mapping argument can take one of the following values: manyToMany, manyToOne, oneToMany or oneToOne. As with the creation of a formal module, the optional last argument controls whether the module is displayed in the user interface after it has been created.

The second form of the perm creates a Link module, similar to the perm Module create(name, description, mapping), but returns error messages instead of generating a run-time DXL error.

close(module)

Declaration

bool close(Module m [,bool save])

Operation

Closes the open module m, with the option of saving changes. If save is true, the user is prompted to save before closing. If save is false, closes the module without saving. If the module is closed, the call fails.
If the operation fails, returns false. If \( m \) is a link module, close only succeeds if there are no loaded linksets and no other module is currently referring to the link module. Any open link modules that \( m \) refers to are also closed.

The Rational DOORS object clipboard is cleared when a module is closed.

Do not access the module handle after the module has been closed.

---

downgrade

Declaration

```dxl
bool downgrade(Module m)
```

Operation

Sets the open mode for module \( m \) to read only, if it is open in edit or shareable mode. This enables other users to open it in shared mode, or one at a time in exclusive edit mode. If the operation succeeds, returns true; otherwise, returns false. If the module is closed, the call fails. If there are unsaved changes to the module, then the user is prompted to save the changes. Alternatively, the save perm can be used prior to downgrade, so that any changes to the module are preserved.

This function is not equivalent to checking whether the current user can modify the given object.

---

downgradeShare

Declaration

```dxl
bool downgradeShare(Module m)
```

Operation

Sets the open mode for module \( m \) to shareable, if it is open in edit mode. This enables other users to open it in shared mode or read mode. If the operation succeeds, returns true; otherwise, returns false. If the module is closed, the call fails. If there are unsaved changes to the module, then the user is prompted to save the changes. Alternatively, the save perm can be used prior to downgrade, so that any changes to the module are preserved.

This function is not equivalent to checking whether the current user can modify the given object.

---

printModule

Declaration

```dxl
void printModule(Module m)
```

Operation

Opens the print dialog box for the open module \( m \).

Example

```dxl
printModule current Module
```
read, edit, share(open module)

**Declaration**

Module read(string name
 [, bool disp[, bool loadStandardView]])

Module edit(string name
 [, bool disp[, bool silent[, bool loadStandardView]]])

Module share(string name
 [, bool disp[, bool silent[, bool loadStandardView]]])

**Operation**

These functions return a module handle for the module named `name`. The name argument can be an absolute or relative path. The `read` function opens the module for reading, `edit` for unshared editing, and `share` for shared editing. The optional `disp` flag enables the visibility of the opened module to be specified; the module is displayed in a window if `disp` is true or omitted.

The optional parameter `silent` specifies whether the user should be prompted when the module cannot be opened in the desired mode because of locks. If this parameter is not supplied it is assumed to be false.

Using the optional parameter `loadStandardView` means you can force the standard view to be loaded as the default. If this parameter is not supplied it is assumed to be false.

**Note:** If a module is open in a particular mode, that same module must not be opened in another mode, if the statement doing this is within a for loop.

**Example**

Module m = edit("/Car/Car user reqts", false)

save(module)

**Declaration**

void save(Module m)

**Operation**

Saves open module `m`.

copy(module)

**Declaration**

bool copy(ModName_ modRef,
       string newName,
       string newDesc)
Operation

Copies module \textit{modRef} to new name \textit{newName}, with description \textit{newDesc}, within the same folder or project. All outgoing links are copied, but incoming links are not copied, and linksets are not updated.

hardDelete(module)

Declaration

\begin{verbatim}
bool hardDelete(ModName_ &modRef)
\end{verbatim}

Operation

Removes module \textit{modRef} from the database (compare with the \textit{softDelete(module)} function); the module cannot be recovered with \textit{undelete(item)} following this operation.

If the operation succeeds, sets the argument to \textit{null}, and returns \textit{true}; otherwise, returns \textit{false}. If the user does not have delete access to the item, or if the module is open, the call fails.

The function \textit{hardDelete} should be used instead of the \textit{delete(item)} function, for all new programs.

\textbf{Note:} \textit{softDelete} must be used on a module before using \textit{hardDelete}.

softDelete(module)

Declaration

\begin{verbatim}
bool softDelete(ModName_ modRef)
\end{verbatim}

Operation

Marks module \textit{modRef} as deleted. The module is not actually deleted until it is purged. Modules marked for deletion can be recovered using the \textit{undelete(item)} function.

When used interactively, a user who tries to use this function on a module with links has to confirm or cancel the operation. In batch mode no confirmation is required.

formalStatus

Declaration

\begin{verbatim}
void formalStatus(Module, String status)
\end{verbatim}

Operation

Displays the supplied string in the third area of the status bar in the specified module, which must be a formal module. If the module is not a formal module a DXL run-time error is generated.
autoIndent

Declaration

bool autoIndent(Module)
void autoIndent(bool)

Operation

The first form returns true if auto-indentation for the main column in the specified module is currently turned on, otherwise it returns false.

The second form sets the auto-indentation status of the current module. The current module should be a formal module, otherwise a run-time DXL error will occur.

Example

print autoIndent current

Module display state

This section defines functions for getting and setting the display attributes of Rational DOORS modules.

level(module get)

Declaration

int level(Module m)

Operation

Returns the display level of module m, which is between 0 (all levels) and 10.

level(module set)

Declaration

void level(int i)

Operation

Sets the display level of the current module. Argument i must be between 0 (all levels) and 10.
Get display state

Declaration

bool filtering(Module m)
bool graphics(Module m)
bool outlining(Module m)
bool showPictures(Module m)
bool showTables(Module m)
bool sorting(Module m)

Operation

Returns the current display state of attributes in open module m: graphics, filtering, outlining, visibility of pictures, visibility of tables, or sorting.

Example

Module m = current
int storeLevel = level m
bool storeGraphics = graphics m
bool storeFiltering = filtering m
bool storeOutlining = outlining m
bool storeSorting = sorting m
functionThatChangesDisplay
// now restore old settings
level storeLevel
graphics storeGraphics
filtering storeFiltering
outlining storeOutlining
sorting storeSorting
if (showTables current) {
    print "table contents are visible"
}
if (!showPictures current) {
    ack "Pictures are not visible"
}
Set display state

Declaration

void filtering(bool onOff)
void graphics(bool onOff)
void linksVisible(bool onOff)
void outlining(bool onOff)
void showPictures(bool onOff)
void showTables(bool onOff)
void sorting(bool onOff)

Operation

Turns on or off in the current module the attributes: filtering, graphics, visibility of links, outlining, visibility of pictures, visibility of tables and sorting.

Example

graphics on
graphics true
graphics off
showPictures true
showTables false

refresh

Declaration

void refresh(Module m)

Operation

Refreshes the display for open module m. Rational DOORS refreshes the current module after the termination of a DXL script. However, scripts that change the displays of other modules, or that create dialog boxes, need to manage display updates explicitly with this function.

bringToFront

Declaration

string bringToFront([Module])
Operation
If a module is supplied it will bring that module window to the front of other windows. If a module is not supplied it will bring the Database Explorer window to the front. Note that this will not bring windows to the front of modal dialogs.

Baselines

This section defines functions that operate on Rational DOORS formal module baselines. The file:
$DOORSHOME/lib/dxl/Example/baseline.dxl
contains a baseline comparison program, which uses the functions described in this section.
Many of the functions use the data type Baseline.

Note: When retrieving information, e.g. annotation, from a baseline you must use them within a for baseline in module loop.

baseline

Declaration
Baseline baseline(int major,
int minor,
string suffix)

Operation
Returns a baseline handle for the combination of the specified major and minor version numbers and suffix string. If the baseline does not have a suffix, use null. This is only used to get a baseline handle for use in the baseline load perm. It cannot be used to retrieve information about that baseline, for example annotation information.

Example
Baseline b = baseline(1, 0, "alpha")

baselineExists

Declaration
bool baselineExists(Module m,
Baseline b)

Operation
Returns true when baseline b exists in module m; otherwise returns false.

Example
print baselineExists(current Module, b)
create(baseline)

**Declaration**

```dxl
void create([Module m,]
            Baseline b,
            string annot)
```

**Operation**

Creates a baseline for module `m` as specified by baseline handle `b` and annotation string `annot`. If the first argument is omitted, it uses the current module.

When this function is used to create a baseline, the module where the baseline is being created will be closed.

Use the `nextMajor`, `nextMinor` functions to instantiate the baseline handle.

delete(baseline)

**Declaration**

```dxl
void delete([Module m,]
            Baseline b)
```

**Operation**

This enables deletion of baselines in formal modules. The first argument defaults to the current module.

**Example**

```dxl
Baseline b = baseline(0, 1, "")
if (baselineExists(current Module, b)) delete(b)
```

Get baseline data

**Declaration**

```dxl
int major(Baseline b)
int minor(Baseline b)
string suffix(Baseline b)
string annotation(Baseline b)
string user(Baseline b)
Date dateOf(Baseline b)
```

**Operation**

These functions return the various data fields associated with baseline `b`. All these functions are included in the “Baselines example program,” on page 309. They must be used within a `for baseline in module` loop.
getMostRecentBaseline

**Declaration**

Baseline
getMostRecentBaseline(Module m
[,bool lastbaseline])

**Operation**

Returns the last baseline. If lastbaseline is set to true, it returns the version number of the last baseline even if it has been deleted. Otherwise, it returns the last baseline that still exists.

**Example**

Module m = current
Baseline b = getMostRecentBaseline(m)
print(major b) "." (minor b) (suffix b) " 
" (annotation b) "n"

getInvalidCharInSuffix

**Declaration**

char getInvalidCharInSuffix(string s)

**Operation**

Returns any character in string s that would be invalid in a baseline suffix.

load

**Declaration**

Module load([Module m,]
Baseline b,
bool display)

**Operation**

Loads baseline b of module m; and if the last argument is on or true, displays it. If the first argument is omitted, it uses the current module.

**Example**

This example loads baseline 1.0 (without a suffix) of the current module, without displaying it.
load(baseline(1,0,null), false)
**nextMajor, nextMinor**

**Declaration**
Baseline nextMajor([string suffix])
Baseline nextMinor([string suffix])

**Operation**
Returns the next major or minor baseline, with or without a suffix.

**Example**
create(nextMajor, "alpha review")
create(nextMajor "A", "alpha review")

**suffix**

**Declaration**
Baseline suffix(string suffix)

**Operation**
Returns a new suffix version of the last baseline.
Can be used to baseline handle for the current version of a module.

**Example**
create(suffix "AA", "no annotation")

**for baseline in module**

**Syntax**
for b in module do {
  ...
}

**where:**
- b is a variable of type Baseline
- module is a variable of type Module

**Operation**
Assigns the baseline b to be each successive baseline found for module module.
Example

Baseline b

for b in current Module do {
    print (major b) "." (minor b) (suffix b) "
    print (user b) "	" (dateOf b) "\n"
    (annotation b) "\n"
}

Baselines example program

// baseline DXL Example
/*
   Example of baseline DXL
*/
Baseline b
Module old = current
for b in current Module do {
    print(major b)"."(minor b)(suffix b)"
    "(annotation b)"\n"
    load(b, true)
    break       // just load first one
}
current = old       // reset
if (confirm "create example baseline?") {
    create(nextMajor, "annotation helps explain
    project history")
    // current Module is closed by create.
}

module(handle)

Declaration

ModName_ module(ModuleVersion modver)

Operation

This returns a handle of type ModName_ for the given ModuleVersion modver. This gives access to information like
name, description, etc. It returns null if the ModuleVersion does not reference an existing module to which the user has
read access.
data(for ModuleVersion)

Declaration

Module data(ModuleVersion modver)

Operation

This returns the data for the given ModuleVersion if the user has it open, loaded into memory. Otherwise, it returns null.

load(ModuleVersion)

Declaration

Module load(ModuleVersion modver, bool display)

Operation

This loads the data (read-only mode) for the given ModuleVersion, if it references a current version or baseline to which the user has read access. If the display argument is true, then the baseline will be displayed. The perm returns the data on success, and null on failure. If the ModuleVersion argument is null, the perm will return null.

moduleVersion(handle)

Declaration

ModuleVersion moduleVersion(Module m)
ModuleVersion moduleVersion(ModName_ modRef[,Baseline b])
ModuleVersion moduleVersion(string index [,Baseline b])

Operation

The first form returns the ModuleVersion reference for the given module version. The module version must be open.
The second form returns the ModuleVersion reference for the given ModName_/Baseline combination. The reference is to the current version of the module if the Baseline argument is omitted.
The third form returns the ModuleVersion reference for the given index/Baseline combination. The reference is to the current version if the Baseline argument is omitted.

isBaseline(ModuleVersion|Module)

Declaration

bool isBaseline(ModuleVersion modver| Module m)
Operation
This returns true if, and only if, the given ModuleVersion or module represents a baseline of a module.

baselineInfo(current Module)

Declaration
Baseline baselineInfo(Module m)

Operation
This returns the baseline designation information of the specified open module m. Returns null if m is a current version.

baseline(ModuleVersion)

Declaration
Baseline baseline(ModuleVersion modver)

Operation
This returns a baseline handle with the major, minor and suffix settings extracted from the ModuleVersion modver supplied as an argument. The user, date and annotation will not be initialized. Returns null if modver corresponds to a current version.

baselineExists(ModuleVersion)

Declaration
bool baselineExists(ModuleVersion modver)

Operation
This returns true if, and only if, the baseline referenced by the ModuleVersion modver argument exists in the database and can be read by the user.

name(ModuleVersion)

Declaration
string name(ModuleVersion modver)

Operation
Returns the name of the module referenced by ModuleVersion modver. Returns null if modver does not refer to a module to which the user has read access.
**fullName(ModuleVersion)**

**Declaration**

```plaintext
string fullName(ModuleVersion modver)
```

**Operation**

Returns the full name, including path, of the module referenced by `ModuleVersion modver`. Returns null if `modver` does not refer to a module to which the user has read access.

---

**versionString(ModuleVersion)**

**Declaration**

```plaintext
string versionString(ModuleVersion modver)
```

**Operation**

Returns the version ID specified in the `ModuleVersion modver`, in the format `<major>.<minor>` where there is no suffix, or `<major>.<minor>(<suffix>)`. If `modver` specifies a current version, this perm returns null.

---

**delete(Baseline)**

**Declaration**

```plaintext
void delete([Module m,] Baseline b)
```

**Operation**

Deletes the specified baseline in a formal module. First argument defaults to the current module.

---

**getMostRecentBaseline(Module)**

**Declaration**

```plaintext
Baseline getMostRecentBaseline(Module m[, bool deleted])
```

**Operation**

Updated the `getMostRecentBaseline` perm to take an optional 2nd argument which if `true` directs the perm to return the version number of the last baseline even if it has been deleted. Otherwise, it returns the last baseline which still exists.
Baseline Set Definition

for BaselineSetDefinition in Folder

Declaration

for baseSetDef in f

where:

    baseSetDef is a variable of type BaselineSetDefinition
    f is a variable of type Folder

Operation

This will return all Baseline Set Definitions baseSetDef whose descriptions are held in the given Folder f, which might also be a Project, to which the user has Read access. The Folder’s Baseline Set Definition list is read from the database at the start of this iterator.

for BaselineSetDefinition in ModName_

Declaration

for baseSetDef in modRef

where:

    baseSetDef is a variable of type BaselineSetDefinition
    modRef is a variable of type ModName_

Operation

This returns all of the Baseline Set Definitions to which the user has Read access, which include the specified module in their lists.

create(BaselineSetDefinition)

Declaration

string create(Folder f, string name, string desc, BaselineSetDefinition &bsd)
Operation
This enables a user with Create access in the Folder to create a new Baseline Set Definition \textit{bsd} with the given name and description. The new Baseline Set Definition will initially inherit its access controls from the folder. The name must conform to the constraints which apply to folder names, and must be unique across the other Baseline Set Definitions in that same folder. The description \textit{desc} might be an empty string.

The newly created Baseline Set Definition is returned in the supplied \textit{bsd} parameter.

The returned string will be non-null in the case that the Baseline Set Definition could not be created:

- If the name clashes with the name of some other Baseline Set Definition on that Folder
- Some i/o or lock error
- Insufficient access

In this case, no Baseline Set Definition will be created (the \textit{bsd} reference will be set to null)

\textbf{rename(BaselineSetDefinition)}

\textbf{Declaration}

\begin{verbatim}
string rename(BaselineSetDefinition \textit{bsd}, string \textit{newName})
\end{verbatim}

\textbf{Operation}

This enables a user with Modify access to change the name of the Baseline Set Definition \textit{bsd}. It returns null on success, and an error message on failure, including insufficient access, or the Baseline Set Definition not being locked for edit, or the name not being unique in that Folder.

\textbf{name(BaselineSetDefinition)}

\textbf{Declaration}

\begin{verbatim}
string name(BaselineSetDefinition \textit{bsd})
\end{verbatim}

\textbf{Operation}

This returns the name of the given Baseline Set Definition \textit{bsd}.

\textbf{setDescription(BaselineSetDefinition)}

\textbf{Declaration}

\begin{verbatim}
string setDescription(BaselineSetDefinition \textit{bsd}, string \textit{desc})
\end{verbatim}

\textbf{Operation}

This enables a user with Modify access to change the description of the Baseline Set Definition. It returns null on success, and an error message on failure, including insufficient access.

A lock on the Baseline Set Definition is required to change the description of that Baseline Set Definition. This lock must be acquired using the lock() perm.
**description(BaselineSetDefinition)**

**Declaration**

```c
string description(BaselineSetDefinition bsd)
```

**Operation**

This returns the description text for the given Baseline Set Definition `bsd`. If the Baseline Set Definition’s information has not been read, this will cause the information to be read from the database.

**for module in BaselineSetDefinition**

**Declaration**

```c
for modRef in bsd do {
    ...
}
```

where:

- `modRef` is a variable of type `ModName_`
- `bsd` is a variable of type `BaselineSetDefinition`

**Operation**

This returns references to all modules (to which the user has Read access) which are included in the Baseline Set Definition `bsd`. If the Baseline Set Definition information has not been read, this will cause the information to be read from the database. Modules that have been deleted (but not purged) are included in the list of modules returned by this iterator.

**addModule(BaselineSetDefinition)**

**Declaration**

```c
string addModule(ModName_ modRef, BaselineSetDefinition bsd)
```

**Operation**

This enables a user with Modify access to add a module to the Baseline Set Definition’s list, if the Baseline Set Definition `bsd` is locked by the user. It will return a string on error, for example if the user does not have Modify access to the Baseline Set Definition or a lock on the Baseline Set Definition.
**removeModule(BaseLineSetDefinition)**

**Declaration**

```plaintext
string removeModule(ModName_ modRef, BaselineSetDefinition bsd)
```

**Operation**

This enables a user with Modify access to remove a module from the Baseline Set Definition’s list, if the Baseline Set Definition `bsd` is locked by the user.

**delete(BaselineSetDefinition)**

**Declaration**

```plaintext
string delete(BaselineSetDefinition &bsd)
```

**Operation**

This enables a user with Delete access to delete a Baseline Set Definition from its parent folder. Once a Baseline Set Definition has been deleted, it cannot be undeleted. On success, the argument Baseline Set Definition will be set to null. A Baseline Set Definition cannot be deleted if another user has it locked for editing.

**lock(BaselineSetDefinition)**

**Declaration**

```plaintext
string lock(BaselineSetDefinition bsd)
```

**Operation**

If the user has Modify access to the Baseline Set Definition `bsd`, this places an exclusive editing lock on it, and reads the information on the Baseline Set Definition from the database. It also ensures that there is a share-lock on its parent folder. Only one session can have a lock at any one time on a Baseline Set Definition, and only a session with a lock can save or modify the Baseline Set Definition, or create a Baseline Set from it. A Baseline Set Definition cannot be modified without it being locked.

Moreover, changes will not be saved to the database until and unless the user performs a save (BaselineSetDefinition).

Notice that it is the responsibility of the programmer to call unlock (BaselineSetDefinition) in order to release a Baseline Set Definition lock acquired by lock (BaselineSetDefinition).

**unlock(BaselineSetDefinition)**

**Declaration**

```plaintext
string unlock(BaselineSetDefinition bsd)
```
Operation
This unlocks a locked Baseline Set Definition bsd, and unlocks its parent Folder if that is not held locked for some other reason. If changes have been made and not saved since the Baseline Set Definition was locked, the Baseline Set Definition information will be read again from the database.

save(BaselineSetDefinition)

Declaration
string save(BaselineSetDefinition bsd)

Operation
This saves the user’s Baseline Set Definition information to the database, as long as the user has an editing lock on the Baseline Set Definition. It returns null on success, and an error message on failure.

read(BaselineSetDefinition)

Declaration
string read(BaselineSetDefinition bsd)

Operation
This reads the current Baseline Set Definition bsd information from the database, and does not require a lock.
If the Baseline Set Definition is locked, and unsaved changes have been made to it, those changes will be lost when read() is called.

isanyBaselineSetOpen(BaselineSetDefinition)

Declaration
bool isAnyBaselineSetOpen(BaselineSetDefinition bsd)

Operation
Returns true if the BaselineSetDefinition has an open baseline set associated with it, and false if it does not. A null argument results in a run-time error.

get(BaselineSetDefinition)

Declaration
AccessRec get(BaselineSetDefinition bsd, string user, string &message)
Operation

On success, this returns the access record for the Baseline Set Definition $bsd$ for the specified user. If $user$ is null, the default access will be returned. The $&message$ string is null on success, and set to an error message on failure.

\[\text{inherited(BaselineSetDefinition)}\]

Declaration

\[
\text{string inherited(BaselineSetDefinition $bsd$)}
\]

Operation

This enables the user to set the Baseline Set Definition $bsd$ to inherit its access controls from its parent Folder.

\[\text{specific(BaselineSetDefinition)}\]

Declaration

\[
\text{string specific(BaselineSetDefinition $bsd$)}
\]

Operation

If the Baseline Set Definition $bsd$ has inherited access rights, this gives it specific access rights, with their initial values inherited from its parent Folder.

\[\text{isAccessInherited(BaselineSetDefinition)}\]

Declaration

\[
\text{string isAccessInherited(BaselineSetDefinition $bsd$, bool $&inherited$)}
\]

Operation

This sets the inherited argument $true$ or $false$ depending on whether the Baseline Set Definition’s access rights are inherited. It returns null on success, and an error message on failure.

\[\text{set(BaselineSetDefinition)}\]

Declaration

\[
\text{string set(BaselineSetDefinition $bsd$, Permission $ps$, string user)}
\]

Operation

This sets a specific access permission for a given $user$. If $user$ is null, then it sets a default access permission. It returns null on success, and an error string on failure.
unset(BaselineSetDefinition)

**Declaration**

```
string unset(BaselineSetDefinition bsd, string user)
```

**Operation**

This removes specific access rights for the given `user` on `BaselineSetDefinition bsd`. If `user` is null, then it sets a default access permission. It returns null on success, and an error string on failure.

unsetAll(BaselineSetDefinition)

**Declaration**

```
string unsetAll(BaselineSetDefinition bsd)
```

**Operation**

This removes all specific access rights from the `Baseline Set Definition bsd`. It returns null on success, and an error message on failure.

for access record in Baseline Set Definition

**Declaration**

```
for ar in bsd do {
    ...
}
```

where:

- `ar` is a variable of type `AccessRec`
- `bsd` is a variable of type `BaselineSetDefinition`

**Operation**

This returns all the specific access right records for the specified `Baseline Set Definition`.

for access record in all Baseline Set Definition

**Declaration**

```
for ar in all bsd do {
    ...
}
```
where:

- $ar$ is a variable of type `AccessRec`
- $bsd$ is a variable of type `BaselineSetDefinition`

**Operation**

Iterates over the access records of the applicable ACL for the specified Baseline Set Definition.

**Example 1**

```dxl
void createBSD()

// creates a BSD containing all the Formal modules in the current Folder
{
    BaselineSetDefinition newBSD = null
    string bsdName = (name current Folder) " modules"
    string bsdDesc = "All modules in this folder"
    string errmess
    errmess = create(current Folder, bsdName, bsdDesc, newBSD)
    if (!null errmess)
    {
        errorBox "Unable to create a new Baseline Set Definition: " errmess
        return
    }
    errmess = lock(newBSD)
    if (!null errmess)
    {
        errorBox "Cannot lock new Baseline Set Definition: " errmess
        return
    }
}
```

// Add modules
Item i
ModName_ mod
for i in current Folder do
{
    if (type(i) == "Formal")
```
mod = module(fullName i)
{
    if (!null mod)
    {
        errmess = addModule(mod, newBSD)
        if (!null errmess)
        {
            errorBox "Could not add module " name(mod) ": " errmess
        }
    }
    errmess = save(newBSD)
    if (!null errmess)
    {
        errorBox "Failed to save Definition: " errmess
    }
    unlock(newBSD)
}

createBSD

Example 2
void printBSDs()
// prints a list of Baseline Set Definitions in the current Folder
// and a list of modules in each Baseline Set Definition
{
    BaselineSetDefinition bsd
    for bsd in current Folder do
    {
        print name(bsd) ": " description(bsd) ""
string errmess = read(bsd)
if (!null errmess)
{
    print " [Could not read Definition: " errmess "]
}
else if (isEmpty(bsd))
{
    print " [Empty Baseline Set Definition]
}
else
{
    ModName_ mod
    for mod in bsd do
    {
        print " " (fullName mod) "
    }
}
print ""
printBSDs

Baseline Sets

for BaselineSet in BaselineSetDefinition

Declaration

    for bs in bsd do {
        ...

DXL Reference Manual
where:

$bs$ is a variable of type BaselineSet

$bsd$ is a variable of type BaselineSetDefinition

**Operation**

This returns the Baseline Sets, in order of creation, which have been created from a given Baseline Set Definition.

---

### `isBaselinePresent(BaselineSet)`

**Declaration**

```c
bool isBaselinePresent(BaselineSet bs, ModName_ modRef)
```

**Operation**

This returns `true` if, and only if, a baseline of the module referenced by `modRef` is in the BaselineSet `bs`.

---

### `create(Baseline Set)`

**Declaration**

```c
string create(BaselineSetDefinition bsd, bool major, string suffix, string annotation, BaselineSet &bs)
```

**Operation**

This enables a user with Create access to create a new (Open) Baseline Set `&bs` from the Baseline Set Definition `bsd`. If `major` is `true`, the version of the Baseline Set will be a new major version number; else it will be a new minor version number. This fails if the Baseline Set Definition is not locked by the user, or if there is already an Open baseline set for it.

The returned string will be null on success, with `&bs` assigned to the baseline set so created. Otherwise, the returned string will be non-null and will contain some description of the failure, in this case `&bs` will be set to null.

---

### `major(BaselineSet)`

**Declaration**

```c
int major(BaselineSet bs)
```

**Operation**

This returns the major version number of a Baseline Set `bs`. 
minor(BaselineSet)

Declaration

int minor(BaselineSet bs)

Operation

This returns the minor version number of a Baseline Set bs.

suffix(BaselineSet)

Declaration

string suffix(BaselineSet bs)

Operation

This returns the suffix (might be null) in the version identifier of the Baseline Set bs.

versionID(BaselineSet)

Declaration

string versionID(BaselineSet bs)

Operation

This returns the whole version identifier of the Baseline Set bs in the form major.minor[(suffix)].

annotation(BaselineSet)

Declaration

string annotation(BaselineSet bs)

Operation

This returns the comment annotation which has been stored with a Baseline Set bs.

user(BaselineSet)

Declaration

string user(BaselineSet bs)

Operation

This returns the name of the user who created the Baseline Set bs.
dateOf(BaselineSet)

Declaration
Date dateOf(BaselineSet bs)

Operation
This returns the date/time when the Baseline Set bs was created.

isOpen(BaselineSet)

Declaration
bool isOpen(BaselineSet bs)

Operation
This returns true for an Open Baseline Set bs, and false for a Closed one.

close(baselineSet)

Declaration
string close(BaselineSet bs)

Operation
This closes an Open Baseline Set bs. It requires the user to have a lock on the Baseline Set Definition, and returns null on success, and an error message on failure (e.g. if the Baseline Set is not Open, or the user does not hold a lock on the Baseline Set Definition).

setAnnotation(BaselineSet)

Declaration
string setAnnotation(BaselineSet bs)

Operation
This enables a user with Modify access to the Baseline Set Definition to change the annotation text on an Open Baseline Set bs. It returns null on success, and an error string on failure (e.g. if BaselineSet is Closed). This should fail if the user does not have a lock on the Baseline Set Definition.
addBaselines(BaselineSet)

Declaration
string addBaselines(Skip modList, BaselineSet bs)

Operation
This enables a user with Modify access to the Baseline Set Definition to baseline a set of modules and add the baselines to an Open Baseline Set. The variable modList is a skip list containing values of type modName_. These modules must be included in the Baseline Set Definition which defines the Baseline Set, and must not already be contained in the Baseline Set. It returns null on success, and an error message on failure (e.g. if the BaselineSet is Closed). It fails without creating or adding any baselines if the user cannot add all of them. It fails if the user does not hold a lock on the Baseline Set Definition.

for ModuleVersion in BaselineSet

Declaration
for modver in bs do {
    ...
}
where:
modver is a variable of type ModuleVersion
bs is a variable of type BaselineSet

Operation
This returns references to all of the baselines, to which the user has Read access, in the Baseline Set.

for ModuleVersion in all BaselineSet

Declaration
for modver in all bs do {
    ...
}
where:
modver is a variable of type ModuleVersion
bs is a variable of type BaselineSet
Operation
This returns references to all baselines in the Baseline Set and all modules which could have been included in the Baseline Set, to which the user has Read access, and which have not been purged.

for BaselineSet in ModName_

Declaration
for bs in modRef do {
  ...
}
where:
  bs is a variable of type BaselineSet
  modRef is a variable of type ModName_

Operation
This returns any open Baseline Sets to which the current version of the specified module can currently be baselined.

baselineSet(ModuleVersion)

Declaration
BaselineSet baselineSet(ModuleVersion modver)

Operation
This returns the Baseline Set, if there is one and the user has Read access to it, which contains the given ModuleVersion modver.

Example 1
void printModuleBSDs()
// prints a list of Baseline Set Definitions which include the current Module
// and a list of Baseline Sets created for each Definition
{
  if (null current Module)
  {
    errorBox "This DXL must be run from a current Module."
    return
  }
BaselineSetDefinition bsd
ModName_ mod = module(current Module)
for bsd in mod do
{
    print name(bsd) " : " description(bsd) "\n"

    string errmess = read(bsd)
    if (!null errmess)
    {
        print " [Could not read Definition: " errmess "]"
    }
else if (isNotEmpty(bsd))
    {
        print " [Empty Baseline Set Definition]"
    }
else
    {
        BaselineSet bs
        for bs in bsd do
        {
            print versionID(bs) " : " annotation(bs) ""

            print " Created by " user(bs) " on " dateOf(bs) ""

        ModuleVersion mv
        for mv in bs do
        {
            print " " (fullName mv) " [" (versionString mv) "]"
        }
    }
    }
print "\n"
Example 2

```dxl
void baselineModuleToSets()

// Adds a new baseline of the current module to any open
// Baseline Set that can include it. Creates a new Baseline Set
// for definitions that include the module but do not have an
// open Baseline Set.
{
    if (null current Module)
    {
        errorBox "This DXL must be run from a current Module."
        return
    }

    string errmess
    BaselineSetDefinition bsd
    BaselineSet bs
    ModName_ mod = module(current Module)
    int skipIndex = 0
    Skip moduleSkip = create
    put (moduleSkip, skipIndex++, mod)

    for bsd in mod do
    {
        print ""

        if (!isAnyBaselineSetOpen(bsd))
        {
            print "Creating new Baseline Set: "

            errmess = lock(bsd)
            if (null errmess)
            {
```
errmess = create(bsd, true, "new", "Created by baselineModuleToSets()", bs)
}

if (!null errmess)
{
    print "Failed to create Baseline Set: " errmess 

    continue
}
unlock(bsd)
}
else
{
    for bs in bsd do
    {
        if (isOpen bs)
        {
            break
        }
    }
}

if (isBaselinePresent(bs, mod))
{
    print "Module is already in the Open Baseline Set."
}
else
{
    errmess = addBaselines(moduleSkip, bs)
    if (null errmess)
    {
        print "Added baseline to Baseline Set " versionID(bs)
History

This section defines DXL functions for manipulating history records. Three main data types are introduced:

- **History**
  a history record

- **HistoryType**
  a type of history

- **HistorySession**
  a summary of a module’s session history. Every time a Rational DOORS module is opened in either edit or shareable mode, a session summary is saved. You can access this information using the functions that act on an object of type HistorySession.

You can only access objects of type History and HistoryType using the for history record in type loop.

You can only access an object of type HistorySession using the for history session in module loop.

Constants (history type)

**Declaration**

```dxl
const HistoryType unknown
const HistoryType createType
const HistoryType modifyType
const HistoryType deleteType
const HistoryType createAttr
const HistoryType modifyAttr
const HistoryType deleteAttr
```
const HistoryType createObject
const HistoryType copyObject
const HistoryType moveObject
const HistoryType modifyObject
const HistoryType deleteObject
const HistoryType unDeleteObject
const HistoryType purgeObject
const HistoryType clipCutObject
const HistoryType clipMoveObject
const HistoryType clipCopyObject
const HistoryType createModule
const HistoryType baselineModule
const HistoryType partitionModule
const HistoryType acceptModule
const HistoryType returnModule
const HistoryType rejoinModule
const HistoryType createLink
const HistoryType modifyLink
const HistoryType deleteLink
const HistoryType insertOLE
const HistoryType removeOLE
const HistoryType changeOLE
const HistoryType pasteOLE
const HistoryType cutOLE
const HistoryType readLocked
const HistoryType synchronizeModule
const HistoryType commentModule
const HistoryType commentObject

**Operation**

These constants represent the different types of history record.

---

**Concatenation (history type)**

The space character is the concatenation operator, which is shown as `<space>` in the following syntax:
**History properties**

Properties are available for use in combination with the . (dot) operator to extract information from a history record. Notably, the properties which are available for individual history entry will depend on the type of that entry. The syntax for using the properties is:

```dxl
hr.property
```

where:

- `hr` is a variable of type `History`
- `property` is one of the history properties

The value of `property` can be one of the following:

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrName</td>
<td>attribute name of history record</td>
</tr>
<tr>
<td>author</td>
<td>author of history record</td>
</tr>
<tr>
<td>newPosition</td>
<td>new position of history record</td>
</tr>
<tr>
<td>position</td>
<td>current position of history record</td>
</tr>
<tr>
<td>type</td>
<td>type of history record; this can be one of the values listed in “Constants (history type),” on page 331</td>
</tr>
<tr>
<td>typeName</td>
<td>type name of history record</td>
</tr>
<tr>
<td>targetInitialName</td>
<td>the name of the target module at the time of link creation (only available to the administrator)</td>
</tr>
<tr>
<td>linkInitialName</td>
<td>the name of the link module at the time of link creation (only available to the administrator)</td>
</tr>
<tr>
<td>plainOldValue</td>
<td>plain text version of the old value</td>
</tr>
<tr>
<td>plainNewValue</td>
<td>plain text version of the new value</td>
</tr>
<tr>
<td>plainOldUnicodeValue</td>
<td>plain text version of the old value, but with any Symbol characters converted into the equivalent Unicode characters, so that the value matches the displayed rich text value</td>
</tr>
<tr>
<td>plainNewUnicodeValue</td>
<td>plain text version of the new value, but with any Symbol characters converted into the equivalent Unicode characters, so that the value matches the displayed rich text value</td>
</tr>
</tbody>
</table>
### Date property

<table>
<thead>
<tr>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
</tr>
<tr>
<td>date of history record</td>
</tr>
</tbody>
</table>

### Integer property

<table>
<thead>
<tr>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>absNo</td>
</tr>
<tr>
<td>absolute number of history record</td>
</tr>
<tr>
<td>numberOfObjects</td>
</tr>
<tr>
<td>number of objects in history record</td>
</tr>
<tr>
<td>oldAbsNo</td>
</tr>
<tr>
<td>old absolute number of history record</td>
</tr>
<tr>
<td>sessionNo</td>
</tr>
<tr>
<td>tracks the manipulation of history information</td>
</tr>
<tr>
<td>sourceAbsNo</td>
</tr>
<tr>
<td>the absolute number of the source object</td>
</tr>
<tr>
<td>targetAbsNo</td>
</tr>
<tr>
<td>the absolute number of the target object</td>
</tr>
</tbody>
</table>

### ModuleVersion property

<table>
<thead>
<tr>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>linkVersion</td>
</tr>
<tr>
<td>the version of the link module</td>
</tr>
<tr>
<td>targetVersion</td>
</tr>
<tr>
<td>the version of the target module</td>
</tr>
</tbody>
</table>

### Any appropriate type property

<table>
<thead>
<tr>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>newValue</td>
</tr>
<tr>
<td>new value of user defined attribute</td>
</tr>
<tr>
<td>oldValue</td>
</tr>
<tr>
<td>old value of user defined attribute</td>
</tr>
</tbody>
</table>

### Example

```dxl
print hr.type
print hr.date
print hr.author
print hr.attrName
print hr.typeName
print hr.position
print hr.newPosition
print hr.numberOfObjects
```
print hr.absNo
print hr.oldAbsNo
print hr.sessionNo
Date histDatOld = hr.oldValue
Date histDateNew = hr.newValue

goodStringOf

Declaration

string goodStringOf(HistoryType ht)

Operation

Returns a string to represent the history type ht in the user interface, for example, "Create Object" for the createObject history type.

stringOf(history type)

Declaration

string stringOf(HistoryType ht)

Operation

Returns the history type ht as a string.

print(history type)

Declaration

void print(HistoryType ht)

Operation

Prints the history type ht in the DXL Interaction window’s output pane.

for history record in type

Syntax

for hr in type do {
    
}


where:

- **hr** is a variable of type `History`
- **type** is a variable of type `Module`, object of type `Object`, or a call to the function `top`

**Operation**

Assigns the variable `hr` to be the history records for modules, objects, or top-level items. Top-level items are those module history records that apply to the whole module, not individual objects. The syntax for looping through top-level items is as follows:

```dxl
for hr in top(module) do { ... }
```

where `module` is of type `Module`.

**Example**

This example prints out the `type` of each top level history record of the current module:

```dxl
History h
for h in top current Module do print h.type
```

**number(history session)**

**Declaration**

```dxl
int number(HistorySession hs)
```

**Operation**

Returns an identifier that is unique within the parent module for a particular session, starting from 0.

**when**

**Declaration**

```dxl
Date when(HistorySession hs)
```

**Operation**

Returns the timestamp for a particular session.

**who**

**Declaration**

```dxl
string who(HistorySession hs)
```
Operation
Returns the name of the Rational DOORS user responsible for a particular session (who opened the module).

Baseline(history session)

Declaration
string baseline(HistorySession hs)

Operation
If a baseline was created during a particular session, returns the details in the format version(suffix).

diff(buffer)

Declaration
string diff(Buffer result, Buffer source, Buffer target, string removeMarkup, insertMarkup)
string diff(Buffer result, Buffer source, Buffer target)
string diff(Buffer result, Buffer source, Buffer target, bool fullRTF)

Operation
Computes the annotated difference, or "redlined difference" between source and target.
The result is valid only when a non-null string is returned.
Removals and insertions are annotated by removeMarkup and insertMarkup -- this must be well-formed RTF strings onto which subsequent text might be concatenated. The standard values for these are "\cf1\strike " and "\cf2\ul " (notice the spaces). See diff/3 for a perm which uses these defaults.
Three colors are defined and might be used within these commands: RED, GREEN and BLUE:
\cf1 - RED
\cf2 - GREEN
\cf3 - BLUE
For the third form of the perm, When true, the RTF returned as a result is full RTF (containing the correct RTF header, font table and color table). When false, the returned result is an RTF fragment, suitable for adding or inserting into a full RTF stream.

Example
DB db = create "Show diff"
DBE textbox = richText(db, "stuff", ",", 200, 200, true)
Buffer buff1 = create()
Buffer buff2 = create()
Buffer resBuf = create()
buff1 = "Old Text"
buff2 = "New Text"
diff(resBuf, buff1, buff2, "\cf1\strike ", "\cf3\ul ")
realize db
useRTFColour(textbox, true)
set(textbox, tempStringOf resBuf)
show db

delete resBuf
delete buff1
delete buff2

Example

Buffer one = create
one = "one"
Buffer two = create
two = "two"
Buffer result = create

diff(result, one, two, false)
print stringOf(result) "\n\n"
diff(result, one, two, true)
print stringOf(result)

Output:

{"cf1\strike one}\{cf3\ul two}
Link History

The name of a module level boolean attribute which controls whether history for link creation and deletion is recorded. Used as a normal attribute but with the addition of the reserved keyword.

**Example**

```dxl
const string LINK_HISTORY_ATTRNAME = "Link History"
Module m = current

if (m != null){

    // get the value
    bool linkHistoryBefore = m.(reserved LINK_HISTORY_ATTRNAME)

    // set the value
    m.(reserved LINK_HISTORY_ATTRNAME) = !linkHistoryBefore

    // get the value again
    bool linkHistoryAfter = m.(reserved LINK_HISTORY_ATTRNAME)

    print "Before: " linkHistoryBefore "\n"
    print "After: " linkHistoryAfter "\n"
}
```

---

**lastModifiedTime**

**Declaration**

`Date lastModifiedTime({Module|Object|Link})`

**Operation**

Returns the date the supplied item was last modified, including the time of the modification.
for history session in module

Syntax

for hs in module do {
...
}

where:

hs is a variable of type HistorySession
module is a variable of type Module

Operation

Assigns the variable hs to be each successive history session record for the specified module.

Example

HistorySession hs
// process module
for hs in current Module do
{
    // identifier, date and user
    print number(hs) "", " when(hs) ", " who(hs)
    string sBaseline = baseline(hs)
    // only relevant if baseline info exists
    if (sBaseline != null)
    {
        // baseline name
        print " - "" sBaseline ": "
    }
    print "\n"
}

History example program

// history DXL Example
/*
   Example history DXL program.
   Generate a report of the current Module's history.
*/
// print a brief report of the history record
void print(History h) {
    HistoryType ht = h.type
    print h.author "\t" h.date "\t" ht "\t"
    if (ht == createType ||
        ht == modifyType ||
        ht == deleteType) { // attribute type
        print h.typeName
    } else if (ht == createAttr ||
        ht == modifyAttr ||
        ht == deleteAttr) {
        // attribute definition
        print h.attrName
    } else if (ht == createObject ||
        ht == clipCopyObject ||
        ht == modifyObject) { // object
        print h.absNo
        if (ht == modifyObject) {
            // means an attribute has changed
            string oldV = h.oldValue
            string newV = h.newValue
            print "(" h.attrName ":" oldV " -> "
            newV ")"
        }
    }
    print "\n"
}
// Main program
History h
print "All history\n"
for h in current Module do print h
print "\nHistory for current Object\n"
for h in current Object do print h
print "\nNon object history\n"
for h in top current Module do print h

Descriptive modules

This section defines DXL functions for Rational DOORS descriptive modules.
create(descriptive module)

**Declaration**

Module create(string name,  
  string description,  
  string prefix,  
  int absno,  
  string filename)

**Operation**

Creates a new descriptive module based on a valid module name and an accessible text file.

If the operation succeeds, returns a reference to the new module; otherwise, returns null.

**Example**

Module m = create("Source","source documentation","S",1,"c:\\docs\\source.txt")

markUp

**Declaration**

Object markUp(Object o,  
  int firstchar,  
  int lastchar)

**Operation**

Marks up a range of object text in a descriptive module, as defined by `firstchar` and `lastchar`.

If the operation succeeds, returns a reference to the newly marked up object; otherwise, returns a reference to the unmarked up object.

If `firstchar` is 1 or less, the range begins at the first character.

If `lastchar` is greater than the number of characters in the specified object, the range ends with the last character in the object.

If `firstchar` is greater than the number of characters in the object, or if `lastchar` is less than 1, or less than `firstchar`, the extracted object contains no text.

**Example**

This example marks up the 2nd, 3rd and 4th characters in the current object:

markUp(current Object, 2, 4)
undoMarkUp

Declaration
void undoMarkUp(Object o)

Operation
Changes a descriptive module object o from being a marked up object to being an unmarked up object. If o does not refer to a marked-up object, the function has no effect.

Example
undoMarkUp(current object)

setUpExtraction

Declaration
bool setUpExtraction(Module m, string formal, string link)

Operation
Sets up the descriptive module m for the extraction of marked up objects to the formal module formal, with links between the source objects and the extractions stored in the link module link.

For a successful operation formal must be open in edit mode, and link must be available for editing.

If the operation is successful, it returns true; otherwise, it returns false.

Example
print setUpExtraction(current Module, "Formal mod", "DOORS Links")

extractAfter

Declaration
void extractAfter(Object source)

Operation
Extracts the marked-up object source to a new object after the current object in the formal module as specified by setUpExtraction.

If the extraction has been incorrectly set up, the function displays a run-time error message is displayed and performs no extraction.

If the extraction is successful, the new object in the formal module becomes the current object.
Example
Module desc = create("Desc mod", "descriptive module", "D", 1, "c:\\info.txt")
Object obj=markUp(current Object 2,22)
edit "Formal module"
setUpExtraction(desc, "Formal module", "Link module")
extractAfter(obj)

extractBelow

Declaration
void extractBelow(Object source)

Operation
Performs the same operation as extractAfter, but inserts the new object below the current object in a formal module.

Example
Module desc = create("Desc mod", "descriptive module", "D", 1, "c:\\info.txt")
Object obj=markUp(current Object 2,22)
edit "Formal module"
setUpExtraction(desc, "Formal module", "Link module")
extractBelow(obj)

Recently opened modules

This section defines DXL functions to access and manipulate the list of recently opened modules.

recentModules

recentModules is a new data type representing the list of recently opened modules.

addRecentlyOpenModule(ModuleVersion)

Declaration
void addRecentlyOpenModule(ModuleVersion ModVer)

Operation
Adds an entry into the recently opened modules list for the supplied module version.
addRecentlyOpenModule(string)

Declaration
void addRecentlyOpenModule(string)

Operation
Constructs a module version from the supplied string, then adds an entry in the recently opened modules list for that module version.

removeRecentlyOpenModule(ModuleVersion)

Declaration
void removeRecentlyOpenModule(ModuleVersion ModVer)

Operation
Removes the entry for the supplied module version from the recently opened modules list.

for {string|ModuleVersion} in recentModules

Operation
Loops through the list of recently opened modules and returns the string representing uniqueID, including baseline version string, or ModuleVersion, for each module.

Syntax
for {str|mv} in recentModules do {
    ...
}

where:

str is a variable of type string

mv is a variable of type ModuleVersion

recentModules is the list of recently opened modules

Example

// This example loops through the list of recently opened modules. It checks for // the presence of two modules, if the first is found it is removed, if the //second is found it is added.

ModuleVersion mod1 = moduleVersion("00000023")
ModuleVersion mod2 = moduleVersion("00000021")
ModuleVersion mod

bool found1 = false
bool found2 = false

for mod in recentModules do {
    if (mod == mod1) {
        found1 = true
    } else if (mod == mod2) {
        found2 = true
    }
}

if (found1) {
    removeRecentlyOpenModule mod1
}

if (!found2) {
    addRecentlyOpenModule mod2
}

---

**Module Properties**

**ModuleProperties** is a new data type representing the properties of a module. It consists of type definitions, attribute definitions, and module attribute values. As with object and module types the . (dot) operator can be used to extract attribute value.
**getProperties**

**Declaration**

```java
string getProperties(ModuleVersion mv, ModuleProperties &mp)
```

**Operation**

Loads type definitions, attribute definitions and module attribute values from the specified `ModuleVersion` into the specified `ModuleProperties`.

---

**delete(ModuleProperties)**

**Declaration**

```java
void delete(ModuleProperties&)
```

**Operation**

Deletes the supplied `moduleProperties` structure. If not called after a call to `getProperties`, the memory will only be released after the context is released.

---

**find(attribute definition in ModuleProperties)**

**Declaration**

```java
AttrDef find(ModuleProperties mp, string AttrName)
```

**Operation**

Returns the attribute definition from the specified `ModuleProperties` whose name matches the supplied string.

---

**for string in ModuleProperties**

**Syntax**

```java
for str in modprops do {
    ...
}
```

where:

- `str` is a variable of type `String`
- `modprops` is a variable of type `ModuleProperties`
Operation
Assigns \textit{str} to be the name of each successive module attribute in \textit{modprops}.

\begin{verbatim}
for AttrType in ModuleProperties

Syntax
for \textit{at} in \textit{modprops} do {
...
}
where:

\begin{tabular}{ll}
\textit{at} & is a variable of type AttrType \\
\textit{modprops} & is a variable of type ModuleProperties \\
\end{tabular}

Operation
Assigns \textit{at} to be each successive module attribute type definition in \textit{modprops}.

Example
ModuleProperties \textit{mp} 
ModuleVersion \textit{mv} 
string \textit{mname} = "/My Project/Module1"
string \textit{s}

\textit{mv} = moduleVersion(module \textit{mname})

\textit{string err1} = getProperties \textit{(mv, mp)}

if (!null err1) {
    print err1 "\n"
}

AttrType \textit{at}
print "Module Types: \n"
\end{verbatim}
for at in mp do {
    print "\t - " (at.name) "\n"
}

print "\nModule Attributes: \n"
for s in mp do {
    print "\t - " s " : "
    val = mp.s"
    print val "\n"
}
Chapter 17

Electronic Signatures

This chapter contains the following topics:

- Signature types
- Controlling Electronic Signature ACL
- Electronic Signature Data Manipulation
- Examples

Signature types

struct SignatureInfo {}

A new type representing signature information.

struct SignatureEntry {}

A new type representing individual signatures. A SignatureEntry is aggregated into exactly one SignatureInfo object.

Controlling Electronic Signature ACL

All access control operations operate on the Electronic Signature information that has been read from the database. Therefore, read operations return results reflecting what was in effect when the data was last refreshed from the database. The data is refreshed by calling getSignatureInfo.

Write operations might result in changes to the access controls, but the access control perms do not commit those changes to the database. Instead, the DXL programmer must explicitly save any changes in order for them to be committed.

SignatureInfoSpecifier__ specifier(SignatureInfo)

Declaration

SignatureInfoSpecifier__ specifier(SignatureInfo si)
Operation

This converter has a Ref implementation. It is an interface selector. It is used for getting and setting permissions for users to change the signature label specifier type for a baseline. It uses the same perms that are used for setting permissions to change the SignatureInfo itself (the rest of the signature configuration). The label specifier is an enumerated type defined in the module, which can have values like signed off, rejected, etc.

For example, if you have a SignatureInfo variable, say sigInfo, which has been initialized using getSignatureInfo, to give you a handle on the signature configuration for a particular baseline, then you get access controls on the signature list using:

- string username
- string access
- AccessRec ac = get(sigInfo, username, access)

Access controls on the label specifier can be retrieved using:

- AccessRec ac2 = get(specifier sigInfo, username, access)

### hasPermission(SignatureInfo, Permission)

**Declaration**

```c
bool hasPermission(SignatureInfo si, Permission& p)
```

**Operation**

Returns true if the current user has permission p to the Signatory ACL of the SignatureInfo object si.

### hasPermission(SignatureInfoSpecifier__, Permission)

**Declaration**

```c
bool hasPermission(SignatureInfoSpecifier__ sis, Permission& p)
```

**Operation**

Returns true if the current user has permission p to the Specifier ACL of the SignatureInfo object si. The specifier() perm is used to cast a SignatureInfo object into a SignatureInfoSpecifier__ object.

### hasPermission(string, SignatureInfo, Permission)

**Declaration**

```c
bool hasPermission(string name, SignatureInfo si, Permission& p)
```

**Operation**

Returns true if the string name has permission p to the Signatory ACL of the SignatureInfo object si.
hasPermission(string, SignatureInfoSpecifier__, Permission)

Declaration
bool hasPermission(string name, SignatureInfoSpecifier__ sis, Permission& p)

Operation
Returns true if the string name has permission p to the Specifier ACL of the SignatureInfo object si. The specifier() perm is used to cast a SignatureInfo object into a SignatureInfoSpecifier__ object.

::do(AccessRec&, SignatureInfo, void)

Declaration
void ::do(AccessRec& ar, SignatureInfo si, void)

Operation
Iterator over Signatory ACL of the SignatureInfo object si.

::do(AccessRec&, SignatureInfoSpecifier__, void)

Declaration
void ::do(AccessRec& ar, SignatureInfoSpecifier__ sis, void)

Operation
Iterator over Specifier ACL of the SignatureInfo object si.

set(SignatureInfo, Permission, string name)

Declaration
string set(SignatureInfo si, Permission& p, string name)

Operation
Sets the Signatory ACL so that string name has Permission p.

set(SignatureInfoSpecifier__, Permission, string name)

Declaration
string set(SignatureInfoSpecifier__ sis, Permission& p, string name)
Operation
Sets the Specifier ACL so that string name has Permission p.

unset(SignatureInfo, string name)

Declaration
string unset(SignatureInfo si, string name)

Operation
Sets the Signatory ACL so that string name has the default access.

unset(SignatureInfoSpecifier__, string name)

Declaration
string unset(SignatureInfoSpecifier__ sis, string name)

Operation
Sets the Specifier ACL so that string name has the default access.

unsetAll(SignatureInfo)

Declaration
string unsetAll(SignatureInfo si)

Operation
Sets Signatory ACL so that all agents have the default access.

unsetAll(SignatureInfoSpecifier__)

Declaration
string unsetAll(SignatureInfoSpecifier__ sis)

Operation
Sets the Specifier ACL so that all agents have the default access.

AccessRec get(SignatureInfo, string name, string& error)

Declaration
AccessRec get(SignatureInfo si, string name, string& error)
Operation

Returns the access record from the Signatory ACL for string *name*. Returns a non-null string if there is an error.

Electronic Signature Data Manipulation

getSignatureInfo(SignatureInfo *si, ModName_ *document, int major, int minor, string *suffix)

Declaration

```c
string getSignatureInfo(SignatureInfo si&, ModName_ document, int major, int minor, string suffix)
```

Operation

Returns in *si* (destructively modifying its contents) a signature information object on the specified baseline *document* (module, with version information). In case of error, a non-null string will be returned, otherwise the null string will be returned.

If the baseline does not exist, this generates an error.

If the baseline does exist, a valid SignatureInfo object will be assigned to *si* and populated with data read from the database. The `isConfigured()` method will return `true`. If the baseline does not have a SignatureInfo object associated with it, a new one is created. The `isConfigured()` method returns `false`, and the SignatureInfo will contain some default values which are dependant on the last configuration specified for that module.

If there is signature information contained in the database for this baseline, that data will be read from the database and *si* will then reflect that data, at the time of the call to `getSignatureInfo`. Changes subsequently made to the database by other sessions will not be reflected in *si* until a further call to `getSignatureInfo` is made.

Since this perm destructively modifies the contents of *si*, any changes that have been made to *si* (for example, a call to `setLabelSpecifier`), are lost. Changes to a SignatureInfo object might be committed to the database by the `save` perm.

isBaselineSignatureConfigured(SignatureInfo)

Declaration

```c
bool isBaselineSignatureConfigured(SignatureInfo si)
```

Operation

Returns whether the SignatureInfo has been configured (if signature Access Controls or signatures have been saved for the associated baseline). See `getSignatureInfo()` for more details.

Note:  This perm does not refresh the SignatureInfo object from the database.
getLabelSpecifier(SignatureInfo)

Declaration

string getLabelSpecifier(SignatureInfo si)

Operation

Returns the signature label specifier. Does not refresh the signature information from the database.

setLabelSpecifier(SignatureInfo si, string newLabel)

Declaration

string setLabelSpecifier(SignatureInfo si, string newLabel)

Operation

Sets the signature label specifier of the supplied si to be the supplied newLabel. This might fail and return a non-null error message if the current user does not have modify access conferred by the Specifier ACL.

This change to the label specifier is not committed to the database until the save(SignatureInfo&) method is called.

appendSignatureEntry(SignatureInfo si, string label, string comment)

Declaration

string appendSignatureEntry(SignatureInfo si, string label, string comment)

Operation

Appends the signature of the current user to the database signature information of the baseline associated with si. This perm is only available when there is a user interface. It will return an error string otherwise. It prompts the user to reconfirm their user name and password, and if this reconfirmation is successful, appends and commits this new signature entry to any existing signatures that might be present in the database.

The label argument will be stored with the signature, and might be used to classify the signature. The baseline signature DXL constrains the user to select the label from the enumeration values of the module’s label specifier type.

The labelOptions argument is intended to contain a newline-separated list of labels available to the user at the time of sign off, as enforced by the calling DXL code.

The comment argument is intended to store any comments that the signatory wishes to record with the signature.

This perm returns an error when Rational DOORS is running in batch mode.

A side-effect of this perm is to refresh si (as would getSignatureInfo) so that it reflects the data that has been committed to the database. As a consequence, any SignatureEntry objects derived from si will be invalidated. Also, any non-committed changes to si will be lost (use the save perm to commit changes before appending a signature).

Since this operation refreshes si, it is possible that the right to sign a baseline will be lost due to a change to the Signatory ACL. In this case an error message will be returned.
save(SignatureInfo si, int &code)

**Declaration**

```cpp
string save(SignatureInfo si, int &code)
```

**Operation**

Save signature information `si` to the database. Returns a non-null string if it fails, in which case the value of `code` will be set to indicate the reason for failure.

On success, this perm writes the specified signature information to the database. Any changes that were made to this signature information since it was refreshed (via `getSignatureInfo`) will be committed to the database.

It is not necessary to call `save` in order to commit changes made by calls to `appendSignatureEntry`. This perm commits those changes before it returns.

Changes made to signature information that do require an explicit call to `save()` are:

- `setLabelSpecifier()`
  
  Any change to access controls

**Returned error codes:**

- out of sequence commit
- other error

An out-of-sequence commit code arises when an attempt is made to commit changes based on an out-of-date read of the signature information. The code will be set to "2" in all other failure cases.

A side-effect of this perm is to refresh `si` (as would `getSignatureInfo`) so that it reflects the data that has been committed to the database. As a consequence, any `SignatureEntry` objects derived from `si` will be invalidated.

::do(SignatureEntry&, SignatureInfo, void)

**Declaration**

```cpp
void ::do(SignatureEntry& sigentry, SignatureInfo si, void)
```

**Operation**

Iterator over each signature entry in the `SignatureInfo` object `si`. The signature entries so obtained are read-only.

The entities will be enumerated in the order in which they were appended to the `SignatureInfo`.

**Note:** This order is independent from the stored dates of the entries.

The signature entries so obtained will be invalidated by execution of any of the following perms on the same `SignatureInfo` object:

- `getSignatureInfo`
- `save`
- `appendSignatureEntry`
As a result, these should not be called when `SignatureEntry` objects remain in scope.

### `getUserName(SignatureEntry)`

**Declaration**  
`string getUserName(SignatureEntry sigentry)`

**Operation**  
Returns the signatory’s user name for the given signature entry.

### `getUserFullName`

**Declaration**  
`string getUserFullName(SignatureEntry sigentry)`

**Operation**  
Returns the signatory’s full user name for the given signature entry.

### `getEmail(SignatureEntry)`

**Declaration**  
`string getEmail(SignatureEntry sigentry)`

**Operation**  
Returns the e-mail address of the signatory for the given signature entry.

### `Date getDate(SignatureEntry)`

**Declaration**  
`Date getDate(SignatureEntry sigentry)`

**Operation**  
Returns the signing date for the given signature entry.

**Note:** This function returns the GMT date/time of the signature and, when formatted to a string, will show the signature time in the time zone of the viewer, not of the signatory.
**Date getLocalDate(SignatureEntry)**

**Declaration**

Date getLocalDate(SignatureEntry sigentry)

**Operation**

Returns the signing date of the given signature entry, offset to compensate for the time zones of the signatory and viewer.

**getFormattedLocalDate(SignatureEntry)**

**Declaration**

string getFormattedLocalDate(SignatureEntry sigentry)

**Operation**

Returns a string representing the date and time of the specified signature in the time zone of the signatory, not the current viewer.

**getLabel(SignatureEntry)**

**Declaration**

string getLabel(SignatureEntry sigentry)

**Operation**

Returns the label, if any, for the given signature entry.

**getLabelOptions(SignatureEntry)**

**Declaration**

string getLabelOptions(SignatureEntry sigentry)

**Operation**

Returns a formatted string representing the choices of label entry available to the signatory at the time of signing.

**getComment(SignatureEntry)**

**Declaration**

string getComment(SignatureEntry sigentry)
Operation

Returns the comment contained in a signature entry. This might be the empty string.

---

**allAttributesReadable(SignatureEntry)**

**Declaration**

```c
bool allAttributesReadable(SignatureEntry sigentry)
```

**Operation**

Returns a boolean indicating if the signatory had read access to all attributes on the signed baseline.

---

**getIsValid(SignatureEntry)**

**Declaration**

```c
bool getIsValid(SignatureEntry sigentry)
```

**Operation**

Returns a boolean value indicating whether the signature hash is still valid for the stored signature entry. This might be used to verify the integrity of signature data.

---

**Examples**

**Add a signature to the latest baseline of the current module**

```c
// Example signatures code - add a signature to the latest baseline of the current module.

Baseline thisBaseline = getMostRecentBaseline(current Module)
if (null thisBaseline || (null load(thisBaseline,true)))
{
    warningBox "No baseline available"
    halt
}

DB signatureDB
DBE addTypeChoice, addAddBtn, addCommentsText
```
SignatureInfo sigInfo
int enumCount = 0
int majorVersion = major(thisBaseline)
int minorVersion = minor(thisBaseline)
string suffix = suffix(thisBaseline)

void addAddCB(DBE x)
// DESCRIPTION : Callback for "OK" button on add signature
// dialog. Calls appendSignatureEntry perm to prompt the user
// to re-authenticate.
// On error, presents a warning box to the user.
// RETURNS : void
{
    string labelString = get(addTypeChoice)
    string commentString = get(addCommentsText)
    string optionsString = ""
    int i
    for (i = 0; i < enumCount; i++)
    {
        if (i > 0)
        {
            optionsString = optionsString "\n"
        }
        optionsString = optionsString get(addTypeChoice,i)
    }

    string message =
    appendSignatureEntry(sigInfo,labelString,optionsString,commentString)
    if (!null message)
    {
        warningBox(signatureDB,"Signature not added: " message ")
    }
} // addAddCB
// First, read the SignatureInfo for the baseline.
string message = getSignatureInfo(sigInfo, module(fullName current Module), majorVersion, minorVersion, suffix)
if (!null message)
{
    warningBox("getSignatureInfo failed: " message ")
    halt
}

// Create the dialog to allow the user to select a label and add a comment.
signatureDB = create("Add Signature", styleFixed)
string labelType = getLabelSpecifier(sigInfo)
AttrType at = null
enumCount = 0

// Get current list of labels from the current version of the module
if (!null labelType)
{
    Module currentVersion = read(fullName current Module, false)
    
    if (!null currentVersion)
    {
        at = find(current Module, labelType)
    }
    if (null at)
    {
        warningBox("Cannot find label specifier type \"" labelType \\"\.\")
        halt
    }
    else if (at.type \\"" !== "Enumeration")
    {
        warningBox("Label specifier is not an enumerated type.\")
        halt
    }
else
{
    enumCount = at.size
}
}

string labelChoices[enumCount]

if (enumCount > 0)
{
    // Get alternative labels from the enumerated type.
    int index
    for (index = 0; index < enumCount; index++)
    {
        labelChoices[index] = at.strings[index]
    }
}

// Create the choice element for the user to select a label.
addTypeChoice = choice(signatureDB,"Signature Label: ",labelChoices,0,20,false)
if (enumCount == 0)
{
    inactive addTypeChoice
}

addCommentsText = text(signatureDB,"Comments: ","",400,150,false)
addAddBtn = button(signatureDB,"OK",addAddCB,styleStandardSize)

show signatureDB

list signatures in the latest baseline

// Signatures example code : list signatures in the latest baseline
// of the current module
if (null current Module)
{
    warningBox "Must run from an open module."
    halt
}

Baseline b = getMostRecentBaseline(current Module)
if (null b || (null load(b,true)))
{
    warningBox "No baseline available"
    halt
}
string dummy[] = {}

DB signaturesDB = create("Baseline Signatures Example",styleFixed)
SignatureInfo signatureInfo = null

DBE timeCombo, sigListView, commentText
DBE labelLabel, labelList, closeBtn
string timeChoices[] = {"signatory's","current"}
DBE timeLabel

static int SIGNATORY_COL = 0
static int DATE_COL = 1
static int LABEL_COL = 2

//*******************************************************************************
void listSignature(SignatureEntry sigEntry, int i, bool localTimes)
// DESCRIPTION : adds an entry in the listView for a given signatureEntry
// RETURNS : void
{
    insert(sigListView,i,getUserName(sigEntry),null,iconUser)
    if (localTimes)
{
    set(sigListView,i,DATE_COL,(dateOf intOf getLocalDate(sigEntry)) "")
}
else
{
    set(sigListView,i,DATE_COL,(dateOf intOf getDate(sigEntry)) "")
}
set(sigListView,i,LABEL_COL,getLabel(sigEntry))
}

//****************************************************************************
void refreshListView(void)
// DESCRIPTION : Populates sigListView with the info in signatureInfo
// RETURNS : void
// ERROR CONDITIONS : null signatureInfo - returns without any action
{
    if (null signatureInfo)
    {
        return
    }

    int i = get(timeCombo)
    bool localTimes = (i == 0)

    int entryNumber
    empty sigListView
    entryNumber = 0
    SignatureEntry sigEntry
    for sigEntry in signatureInfo do
    {
        listSignature(sigEntry,entryNumber,localTimes)
        entryNumber++
    }
} // void refreshListView(void)
void closeDB(DB x)
// DESCRIPTION : close function for the signature dialog. Hides it.
// RETURNS : void
{
    hide signaturesDB
    halt
}

void closeDB(DBE x)
// DESCRIPTION : close function for the signature dialog. Hides it.
// RETURNS : void
{
    closeDB(signaturesDB)
}

void refreshSigsDB()
// DESCRIPTION : refreshes the signatures list with the signature info from
//                the database, in the specified baseline.
// RETURNS : void
{
    ModName_ thisModule = module (fullName current Module)

    string message = getSignatureInfo(signatureInfo, thisModule, major(b),
                                       minor(b), suffix(b))

    set(commentText,"")
    set(labelList,"")
    if (!null message)
    {
        warningBox(signaturesDB,"Cannot display signatures for this baseline:
        " message "\nThe baseline signature dialogue will be closed.")
    }
closeDB(signaturesDB)
}
else
{
    refreshListView()
}
} // refreshSigsDB

//***************************************************************
void timeComboCB(DBE x)
// DESCRIPTION : Callback for the time-zone selection combo
// RETURNS : void
{
    refreshListView()
}

//***************************************************************
void sigDeselectCB(DBE x, int selectedEntry)
// DESCRIPTION : Deselect callback for listView - null-op.
// RETURNS : void
{
}

//***************************************************************
void sigSelectCB(DBE x, int selectedEntry)
// DESCRIPTION : Selection callback for signatures list
// RETURNS : void
{
    int indexScan = 0
    SignatureEntry sigEntry
    for sigEntry in signatureInfo do
    {
if (indexScan == selectedEntry)
{
    set(commentText, getComment(sigEntry))
    set(labelList, getLabelOptions(sigEntry))
    break
}
indexScan++
}
} // sigSelectCB

// DEFINE MAIN DIALOG
sigListView = listView(signaturesDB, 0, 405, 8, dummy)
set(sigListView, sigSelectCB, sigDeselectCB, sigSelectCB)
timeCombo=choice(signaturesDB, "Display time at", timeChoices, 0, 9, false)
set(timeCombo, timeComboCB)
beside signaturesDB
timeLabel = label(signaturesDB, "location.")
below signaturesDB
commentText = text(signaturesDB, "Comments: ", ",", 100, 100, true)
labelList = text(signaturesDB, "Available labels: ", ",", 160, 100, true)

// BUTTONS
close(signaturesDB, true, closeDB)

realize signaturesDB
insertColumn(sigListView, SIGNATORY_COL, "Signatory", 150, iconNone)
insertColumn(sigListView, DATE_COL, "Date / Time", 150, iconNone)
insertColumn(sigListView, LABEL_COL, "Label", 100, iconNone)

refreshSigsDB()
show signaturesDB
Chapter 18

Objects

This chapter describes features that operate on Rational DOORS objects:

• About objects
• Object access controls
• Finding objects
• Current object
• Navigation from an object
• Object management
• Information about objects
• Selecting objects
• Object searching
• Miscellaneous object functions

About objects

Functions manipulate Rational DOORS objects via the `object` data type. An important property of a Rational DOORS formal module is that the objects within the module are structured as a tree; the functions for creating and navigating objects therefore use the following tree terminology:

- **parent**: the object immediately above an object
- **child**: any object immediately below an object
- **sibling**: any object that shares a parent with another object

Object DXL can be found in nearly every example DXL program given in this manual or in the DXL library.

Object access controls

This section describes functions that report on access rights for an object. For all except the `canRead(object)` function, the module must be open for exclusive edit.
canCreate(object)

**Declaration**

```cpp
bool canCreate(Object o)
```

**Operation**

Returns `true` if the current Rational DOORS user has create access to object `o`; otherwise, returns `false`.

canControl(object)

**Declaration**

```cpp
bool canControl(Object o)
```

**Operation**

Returns `true` if the current Rational DOORS user can change the access controls on object `o`; otherwise, returns `false`.

canRead(object)

**Declaration**

```cpp
bool canRead(Object o)
```

**Operation**

Returns `true` if the current Rational DOORS user can read object `o`; otherwise, returns `false`.

canModify(object)

**Declaration**

```cpp
bool canModify(Object o)
```

**Operation**

Returns `true` if the current Rational DOORS user can modify object `o`; otherwise, returns `false`.

canDelete(object)

**Declaration**

```cpp
bool canDelete(Object o)
```

**Operation**

Returns `true` if the current Rational DOORS user can delete object `o`; otherwise, returns `false`. 
canLock(object)

Declaration

bool canLock(Object o)

Operation

Returns true if the current Rational DOORS user can lock object o, which must be in a lockable section. It returns false for the following conditions:

• o is null
• o is contained within a module that is currently open read-only
• o is contained within a module that is currently open for exclusive edit
• o is not contained within a formal module
• the user does not have create or modify access to the object at the start of o’s editable section

canUnlock(object)

Declaration

bool canUnlock(Object o)

Operation

Returns true if the current Rational DOORS user can unlock object o, which must be in a lockable section. It returns false for the following conditions:

• o is null
• o is contained within a module that is currently open read-only
• o is contained within a module that is currently open for exclusive edit
• o is not contained within a formal module
• the user does not currently have o locked

Finding objects

This section defines functions that allow DXL programs to navigate through the objects in a module.

object(absno)

Declaration

Object object(int absno[,Module m])
Operation
Returns the object with the specified absolute number. If no module argument is supplied, the current module is searched.

all

This function is used in a for loop operating on modules, as shown in the following syntax:
all(Module module)
Returns a handle for module (see the for object in all loop).

document

This function is used in a for loop operating on modules, as shown in the following syntax:
document(Module module)
Returns a handle for module (see the for object in document loop).

entire

This function is used in a for loop operating on modules, as shown in the following syntax:
entire(Module module)
Returns a handle for module (see the for object in entire loop).

module(containing object)

Declaration
Module module(Object o)

Operation
Returns the module that contains object o.

top

This function is used in for loops operating on projects and modules, as shown in the following syntax:
top(Module module)
Returns a handle for module (see the loops for history record in type and for object in top).
for object in all

Syntax

for o in all(module) do {
    ...
}

where:

- **o** is a variable of type Object
- **module** is a variable of type Module

Operation

Assigns the variable o to be each successive object in module. It includes table and row header objects and the cells, unless a filter is defined.

This loop respects the current display set; an object is only returned if it is displayed under the current filter, level setting, and so on. However, table header objects are always accounted for, even if the table is not visible in the current display due to a filter.

Deleted objects are included when they are visible and excluded when they are not visible. This is the case for all deleted objects except deleted table header objects, which are always displayed. Object numbering depends on whether deleted objects are displayed. If they are displayed, they are numbered. If they are not displayed, they are not numbered.

Example

Object o

for o in all current Module do {
    print identifier o "\n"
}

for object in entire

Syntax

for o in entire(module) do {
    ...
}

where:

- **o** is a variable of type Object
- **module** is a variable of type Module
Operation

Assigns the variable \( o \) to be each successive object in \( module \) regardless of its deleted state or the current display set. It includes table and row header objects and the cells.

for object in document

Syntax

\[
\text{for } o \text{ in document}(\text{module}) \text{ do }
\]
\[
\ldots
\]
\[
\}
\]

where:

\( o \) is a variable of type \text{Object}  \\
\( module \) is a variable of type \text{Module}

Operation

Assigns the variable \( o \) to be each successive object in \( module \). It is equivalent to the \text{for object in module} loop, except that it includes table header objects, but not the row header objects nor cells.

Example

Object \( o \)

\[
\text{for } o \text{ in document current Module do }
\]
\[
\quad \text{print identifier } o \text{ "\n"
\}
\]

for object in module

Syntax

\[
\text{for } o \text{ in } \text{module} \text{ do }
\]
\[
\ldots
\]
\[
\}
\]

where:

\( o \) is a variable of type \text{Object}  \\
\( module \) is a variable of type \text{Module}

Operation

Assigns the variable \( o \) to be each successive object in \( module \) in depth first order, including the cells only of any Rational DOORS native tables. Depth first order is the order in which objects are displayed down the page in a formal module.
This loop respects the current display set; an object is only returned if it is displayed under the current filter, level setting, and so on. Deleted objects are included when they are displayed and excluded when they are not displayed. Object numbering depends on whether deleted objects are displayed. If they are displayed, they are numbered. If they are not displayed, they are not numbered.

**Example**

```dxl
Object o

for o in (current Module) do
    print (o."Object Heading") "\n"
```

### for object in object

**Syntax**

```dxl
for o in parent do {
    ...
}
```

where:

- `o` is a variable of type `Object`
- `parent` is an object of type `Object`

Assigns `o` to each successive child of object `parent`.

This loop ignores filters, such that even if objects are filtered, they are still returned by this function. Deleted objects are included when they are displayed and excluded when they are not displayed.

**Example**

```dxl
Object o
Object po = current

for o in po do {
    print (o."Object Heading") " is a child of "
    print (po."Object Heading") "\n"
}
```

### for object in top

**Syntax**

```dxl
for o in top(module) do {
    ...
}
```
where:

\[ o \quad \text{is a variable of type Object} \]

\[ module \quad \text{is a variable of type Module} \]

Assigns \( o \) to each successive top-level object in \( module \), including table headers. Top-level objects are those at level 1. This function accesses all top level objects regardless of the current display set, which is different from the \( \text{for object in module} \) loop. Deleted objects are included, if they are displayed. Object numbering depends on whether deleted objects are displayed. If they are displayed, they are numbered. If they are not displayed, they are not numbered.

**Example**

Object \( o \)
Module \( m = \text{current} \)
for \( o \) in \( \text{top} m \) do {
  print \( o \)."Created On" "\n"
}

---

**Current object**

This section defines functions that are concerned with getting or setting the current object in a Rational DOORS module.

**Setting current object**

The assignment operator = can be used as shown in the following syntax:

\[ \text{current} = \text{Object} \ \text{object} \]

Makes \( object \) the current object. See also, the \( \text{current}(\text{object}) \) function.

For large DXL programs, when you set the current object, cast the current on the left hand side of the assignment to the correct type. This speeds up the parsing of the DXL program, so when your program is first run, it is loaded into memory quicker. It does not affect the subsequent execution of your program. So:

\[ \text{current} = \text{newCurrentObject} \]

becomes

\[ (\text{current ObjectRef__}) = \text{newCurrentObject} \]

Note that this cast only works for assignments to \( \text{current} \). It is not useful for comparisons or getting the value of the current object.

**Example**

\[ \text{current} = \text{first current Module} \]
\[ \text{current} = \text{below current} \]
current = create last below current

current(object)

**Declaration**

Object current([Module m])

**Operation**

Returns a reference to the current object of module m, or the current module if m is omitted.

**Example**

Object o = current
Module m = edit "Car user reqts"
Object o = current m

---

**Navigation from an object**

This section defines functions that allow navigation across a Rational DOORS module relative to a given object.

**Specific object**

The index notation, [ ], can be used to find a specific object, as shown in the following syntax:

Object o[int n]
Module m[int n]

This returns the n-th child of object o counting from 1, or the n-th top-level child of module m, counting from 1.

**gotoObject**

**Declaration**

Object gotoObject(int absno, Module m)

**Operation**

Changes the display of the specified module so that the object with the specified absolute number is brought into the display, and made current. This perm will change the current view in order to ensure that this object can be displayed.

Returns the Object with that absolute number.
Vertical navigation

**Declaration**

Object first(Object o)
Object last(Object o)
Object next(Object o)
Object parent(Object o)
Object previous(Object o)
Object first(Module m)
Object last(Module m)

**Operation**

The first five functions take an object argument o, and return an object, which is the object in the position relative to o as stated by the function:

- **first** returns the first child of object o
- **last** returns the last child of object o
- **parent** returns the parent of object o
- **previous** returns the previous object from object o in a depth first tree search (the same order as for o in module do)
- **next** returns the next object from object o in a depth first tree search (the same order as for o in module do)

If navigation is attempted to somewhere where no object exists, returns null. These functions are used for vertical navigation of a Rational DOORS module.

The last two functions return the first and last objects of module m in a depth first tree search, that is the first and last objects as they appear in a displayed module.

**Example**

This example finds objects relative to the passed object argument:

Object o = current
Object co = first o
if (null co) {
    print "Current object has no children.\n"
} else {
    if ((last o) == co) {
        print "current has one child: " (o."Object Heading") "\n"
    }
}
print (identifier o) " == " (identifier parent co) "\n"

if (null o[3])
    print "current object does not have 3rd child\n"
if (null previous o)
    print "Current object is first in module.\n"
if (null next o)
    print "Current object is last in module.\n"
if (!null next o) {
    Object here = previous next o
    print (identifier o) " and " (identifier here) " are the same\n"
}

This example finds objects in the current module:
Object o1 = first current Module
Object o2 = last current Module
int count = 1
while (o1 != o2) {
    count++
    o1 = next o1
}
print count " objects displayed in module\n"
o1 = (current Module)[3]
// get 3rd top level object
print identifier o1

---

**Horizontal navigation**

These functions are similar to the vertical navigation functions, but take as an argument a call to the function sibling, which returns a handle to allow navigation between sibling objects (children of the same parent).

**Declaration**

Object first(sibling(Object o))
Object last(sibling(Object o))
Object next(sibling(Object o))
Object previous(sibling(Object o))
Operation

These functions return an object at the current level of hierarchy: `first sibling` and `last sibling` return the first and last objects. Function `first sibling` works with the current display set, so hierarchies might disappear as the display set changes during navigation.

The functions are used for horizontal navigation of a Rational DOORS module.

Example

```dxl
Object o = current
Object po = parent o
if ((null previous sibling o) &&
    (null next sibling o)) {
    print (o."Object Heading") " is the only
    child of " // -(po."Object Heading") "\n"
    print "and " (identifier first sibling o) " ==
    " // -(identifier first sibling o) " ==
    " (identifier o) "\n"
}
```

Object management

This section defines the functions for creating, moving and deleting objects.

**Note:** The creation of tables, table rows, columns and cells is handled by special-purpose functions, which are described in “Tables,” on page 817.

create(object)

**Declaration**

```dxl
Object create(Module m)
Object create(Object o)
Object create(after(Object o))
Object create(before(Object o))
Object create(below(Object o))
object create(first(below(Object o)))
Object create(last(below(Object o)))
```
Operation

These functions create an object, whose position is controlled by the argument passed to the function, as follows:

<table>
<thead>
<tr>
<th>Argument syntax</th>
<th>New object is</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module ( m )</td>
<td>The first object in module ( m ); any existing objects at level 1 are moved after the new object</td>
</tr>
<tr>
<td>Object ( o )</td>
<td>At the same level and immediately after the object ( o )</td>
</tr>
<tr>
<td>after(Object ( o ))</td>
<td>At the same level and immediately after the object ( o ) (same as without after)</td>
</tr>
<tr>
<td>below(Object ( o ))</td>
<td>The first child of object ( o )</td>
</tr>
<tr>
<td>first(below(Object ( o )))</td>
<td>The first child of object ( o ) (same as without first)</td>
</tr>
<tr>
<td>last(below(Object ( o )))</td>
<td>The last child of object ( o )</td>
</tr>
</tbody>
</table>

In each case, the function returns the created object.

Example

This example creates newo at the same level and immediately after \( o \).

Object \( o \) = current
Object newo = create \( o \)

which is equivalent to:

Object \( o \) = current
Object newo = create after \( o \)

This example creates newo at the same level and immediately before \( o \).

Object \( o \) = current
Object newo = create before \( o \)

This example creates newo as the first child of \( o \).

Object \( o \) = current
Object newo = create below \( o \)

which is equivalent to:

Object \( o \) = current
Object newo = create first below \( o \)

This example creates newo as the last child of \( o \).

Object \( o \) = current
\( o \) = create last below \( o \)
move(object)

Declaration

void move(Object o1,
    Object o2)
void move(Object o1,
    below(Object o2))
void move(Object o1,
    last(below(Object o2)))

Operation

These functions move an object to a position, which is controlled by the second argument passed to the function, as follows:

<table>
<thead>
<tr>
<th>Argument syntax</th>
<th>Moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object o2</td>
<td>object o1 and its descendants to be immediately after object o2</td>
</tr>
<tr>
<td>below(Object o2)</td>
<td>object o1 and its descendants to be the first child below o2</td>
</tr>
<tr>
<td>last(below(Object o2))</td>
<td>object o1 and its descendants to be the last child below o2</td>
</tr>
</tbody>
</table>

Example

This example moves the last object in the module to be the first child of the first object:

Object p = first current Module
Object o = last current Module
move (o, below p)

This example moves the last object in the module to be the last child of the first object:

Object p = first current Module
Object o = last current Module
move(o, last below p)

canDelete

Declaration

string canDelete(Object o)
Operation

Returns null if object o can be deleted; otherwise returns a string "object has descendants". The softDelete(object) function works on an object that has descendants.

flushDeletions

Declaration

void flushDeletions()

Operation

Flushes any deletions performed by a DXL program. Normally Rational DOORS structures are only marked for deletion when the DXL program exits; this command makes any pending deletions happen immediately. Do not flush deletions inside a for loop, because the loop might depend on the presence of an object.

hardDelete(object)

Declaration

void hardDelete(Object o)
void delete(Object o)

Operation

Removes object o; the object cannot be recovered with undelete following this operation. If the operation fails, returns an error message (see also the canDelete function).

The form delete is provided for backwards compatibility only. The function hardDelete should be used for all new programs.

sectionNeedsSaved

Declaration

bool sectionNeedsSaved(Object o)

Operation

Returns true if o is contained within an object hierarchy that has been modified but not saved. Otherwise, returns false.

softDelete(object)

Declaration

void softDelete(Object o[, bool checkLinks])
Operation
Marks object as deleted. The object is not actually deleted until it is purged. Objects marked for deletion can be recovered using the undelete(object) function. If the optional argument checkLinks is set to true, then an error will be given if any of the objects children have incoming links.

undelete(object)

Declaration
string undelete(Object o)

Operation
Restores object o. On success returns null. On error, the error condition is returned to the user.

purgeObjects_

Declaration
string purgeObjects_ (Module mod)

Operation
Removes all soft deleted objects from module mod. Once executed, these objects cannot be recovered. The name ends in '_' to discourage casual use.

purgeObject_

Declaration
string purgeObject_ (Object o)

Operation
Removes the specified soft-deleted object. Once executed, this object cannot be recovered. The name ends in '_' to discourage casual use.

Information about objects

This section defines functions that return information about objects.
Object status

Declaration

bool canRead(Object o)
bool canWrite(Object o)
bool leaf(Object o)
bool isDeleted(Object o)
bool isFiltered(Object o)
bool isOutline(Object o)
bool isSelected(Object o)
bool isVisible(Object o)
bool modified(Object o)

Operation

Each function returns true for a condition that is defined by the function name:

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns true if</th>
</tr>
</thead>
<tbody>
<tr>
<td>canRead</td>
<td>the user has read access to object o</td>
</tr>
<tr>
<td>canWrite</td>
<td>the user has write access to object o</td>
</tr>
<tr>
<td>leaf</td>
<td>object o has no children, or has children objects that are deleted, but not displayed</td>
</tr>
<tr>
<td>isDeleted</td>
<td>object o has been soft deleted</td>
</tr>
<tr>
<td>isFiltered</td>
<td>object o is accepted in the current filter</td>
</tr>
<tr>
<td>isOutline</td>
<td>object o would appear in outline mode</td>
</tr>
<tr>
<td>isSelected</td>
<td>object o is selected</td>
</tr>
<tr>
<td>isVisible</td>
<td>object o is part of the current display set</td>
</tr>
<tr>
<td>modified</td>
<td>object o has been modified since the last baseline of the module</td>
</tr>
</tbody>
</table>

getColumnBottom

Declaration

Object getColumnBottom(Object o)
Operation
Returns the bottom cell of the table column that contains o; otherwise, returns null.

columnTop

Declaration
Object getColumnTop(Object o)

Operation
Returns the top cell of the table column that contains o; otherwise, returns null.

level

Declaration
int level(Object o)

Operation
Returns the object level of object o. Level 1 is the top level of the module.

identifier

Declaration
string identifier(Object o)

Operation
Returns the identifier, which is a combination of absolute number and module prefix, of object o as a string.

number

Declaration
string number(Object o)

Operation
Returns the hierarchical object number (for example 2.1.1-0.1) of object o as a string.

Selecting objects

This section defines functions concerned with selecting objects.
getSelection

Declaration

Object getSelection(Module m)
void getSelection(Module m,
    Object &start,
    Object &finish)

Operation

The first form gets the first object of a selection in module \( m \).
The second form gets the current selection in module \( m \), and sets object variables \( start \) and \( finish \) to the beginning and end of it.
The start and end objects must be siblings.

setSelection

Declaration

void setSelection(Object o)
void setSelection(Object start,
    Object finish)

Operation

The first form makes object \( o \) the start and finish of the current selection.
The second form sets the selection in the current module to begin at object \( start \) and end at object \( finish \).
The start and end objects must be siblings.

deselect

Declaration

void deselect(Object o)
void deselect(Module m)

Operation

Deselects object \( o \) or the current selection in module \( m \).
Object searching

This section defines functions that are used by Find/Replace when highlighting an object, or an object’s attribute.

**setSearchObject**

**Declaration**

```java
void setSearchObject(Object, int columnIndex)
```

**Operation**

Used by Find/Replace to mark either a specific attribute of the object in a column by surrounding it in a colored box (the same color as an outgoing link). This indicates which specific part of the object has been matched by the find operation. If no valid/visible column is supplied, the object is marked by lines above and below the entire object.

**Example**

```java
Object o = object(4)
int mainColumn = 1
setSearchObject(o, mainColumn)
```

**getSearchObject**

**Declaration**

```java
Object getSearchObject(Module, int &columnIndex)
```

**Operation**

Returns the object and column number of the highlighted attribute in the given module.

**Example**

```java
Module m = current
int col
Object o = getSearchObject(m, col)
```

**clearSearchObject**

**Declaration**

```java
void clearSearchObject(Object)
void clearSearchObject(Module)
```
Operation
Clears the highlighting put in place by setSearchObject. Currently, if an object is provided, that object need not be the highlighted object, but this could change.

Example
Object o = current
clearSearchObject(o)

highlightText

Declaration
bool highlightText(Object, int start, int stop, int colIndex, bool isHeading)

Operation
Highlights text in the given module, in the given column from cursor position start to cursor position stop.

Example
//Highlights the first 10 characters of the current objects heading
highlightText(current Object, 10, 20, 1, true)

getInPlaceColumnIndex

Declaration
int getInPlaceColumnIndex(Module)

Operation
Returns the column index where in-place editing is taking place.

Miscellaneous object functions

This section defines functions that affect the display of an object or use the clipboard.

inplaceEditing

Declaration
bool inplaceEditing(Module m)
Operation

This returns true if the module \( m \) is a formal module which is currently displayed and in-place edit mode is activated for a displayed attribute.

---

Object

Declaration

Object object(int \( i \), Module \( m \))

Operation

Returns the object with the specified absolute number. If no Module argument is supplied, the current module is searched.

Example

Object \( o \) = object(4)
print identifier \( o \)

---

Clipboard general functions

Declaration

bool cut()
bool copyFlat()
bool copyHier()
bool pasteSame()
bool pasteDown()
bool clearClipboard()
bool clipboardIsEmpty()
bool clipboardIsTransient()

Operation

Each function performs an action or status check defined by the function name as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>cut</td>
<td>Cuts the current object and all of its children, and stores them on the clipboard. If the operation succeeds, returns true; otherwise, returns false.</td>
</tr>
<tr>
<td>copyFlat</td>
<td>Copies the current object to the clipboard. If the operation succeeds, returns true; otherwise, returns false.</td>
</tr>
</tbody>
</table>
### splitHeadingAndText

**Declaration**

```dxl
string splitHeadingAndText(Object)
```

**Operation**

Splits the Object Heading and Object Text of the given object. The heading will be moved to a new object, and the heading of the given object will be emptied. The given object will be demoted to become the first child of the new object. Returns a null string on success or an error message on failure.

**Example**

```dxl
Object o = current
string s = splitHeadingAndText(o)
if (null s){
   print "Object split successfully."
} else {
   print "Error splitting object : " s
}
```
getCursorPosition

Declaration

```c
int getCursorPosition(Module, bool &isHeading)
```

Operation

If no attributes in the given module are activated for in-place editing then -1 is returned. Otherwise it returns the position of the cursor in the attribute currently being edited, if that attribute is the Object Heading then `isHeading` will be set to true, otherwise it will be set to false.

Example

```c
bool isHeading

print getCursorPosition(current Module, isHeading) ""
```
Chapter 19

Links

This chapter describes features that operate on Rational DOORS links:

- About links and link module descriptors
- Link creation
- Link access control
- Finding links
- Versioned links
- Link management
- Default link module
- Linksets
- External Links
- OSLC Link Discovery
- Rational DOORS URLs

About links and link module descriptors

The underlying database architecture of Rational DOORS links affects the way in which link DXL must be written. Link modules store linksets, not actual links. Link modules can be placed in any folder in the hierarchy except the database root folder, but they are normally placed in the folder containing the source module.

Links are stored in the module corresponding to the source of the link. This means that the user must have write permission in the source module to create or modify a link.

This causes an asymmetry in DXL programs that handle links. Any code trying to access an incoming link must have the source module loaded. Outgoing links are always immediately available in a formal module. However, the target module might not be open, in which case the target function returns null.

Rational DOORS links are represented in DXL in by the Link data type.

A folder or project can specify the link modules to be used when a link is created between a pair of modules, the source of which is in the folder. This source/target module pairing is called a link module descriptor, which is represented by the LinkModuleDescriptor data type.

Note: To obtain a type LinkModuleDescriptor handle, you must use the for link module descriptor in folder loop.
Each pairing contains the name of the link module, a description, and a boolean flag overridable. The overridable flag specifies whether that link module must be used for links between the specified source and target module. If overridable is false, newly created links must be in that link module; specifying a different link module at the time a link is created causes a run-time error. If overridable is true, you can specify a different link module. The modules referenced in the link module descriptor might but need not already exist at the time the link module is specified.

---

**Link creation**

This section defines the operators used to create links.

**Link operators**

Two operators create links, as shown in the following syntax:

Object source -> [string linkModuleName ->] Object target
Object target <- [string linkModuleName <-] Object source

The -> operator creates an outgoing link from object source to object target via link module linkModuleName. If linkModuleName is omitted the link goes via the default link module (see “Default link module,” on page 410.

The <- operator creates an incoming link from object source to object target via link module linkModuleName. If linkModuleName is omitted the link goes via the default link module.

These operators are also used in the for loops defined in “Finding links,” on page 395.

**Example**

This example creates a link from the current object of the current module to the first object of module target via the link module tested by.

(current Object) -> "tested by" -> (first read "target")

This example creates a link to the current object of the current module from the first object of module source via the link module tested by. Because links are stored in the source module, you must open source for editing to allow the link to be created.

(current Object) <- "tested by" <- (first edit "source")

---

**Link access control**

This section describes a function that reports on access rights for links.
canDelete(link)

Declaration

bool canDelete(Link l)
string canDelete(Link l)

Operation

The first form returns true if the current Rational DOORS user can delete link l. Otherwise, returns false.
The second form returns a null string if the current Rational DOORS user can delete link l. Otherwise, it returns an error message.

Finding links

This section defines for loops that allow DXL programs to navigate through the links in a module. Links are referred to by the Link or LinkRef data type.

for all outgoing links

Syntax

for outLink in Object srcObject ->(string linkModuleName) do {
  ...
}

where:

  outLink                is a variable of type Link
  srcObject             is a variable of type Object
  linkModuleName        is a string variable

Operation

Assigns the variable outLink to be each successive outgoing link from object srcObject via link module named linkModuleName. The string linkModuleName can be a specific link module name, or the string "*" meaning any link module.

Example

Link l
for l in (current Object) -> "*" do {
    string user = l."Created By"
    print user "\n"
}

for all incoming links

Syntax

for inLink in Object tgtObject<-(string linkModuleName) do {
    ...
} where:

   inLink is a variable of type Link or LinkRef
   tgtObject is a variable of type Object
   linkModuleName is a string variable

Operation

Assigns the variable inLink to be each successive incoming link arriving at object tgtObject via link module named linkModuleName. The string linkModuleName can be a specific link module name, or the string "*" meaning any link module.

Note: This loop only assigns to inLink incoming link values for which the source object is loaded; unloaded links are not detected.

Example

Link l
for l in current Object<="*" do {
    string user = l."Created By"
    print user "\n"
}

for each incoming link

Syntax

for LinkRef in Object tgtObject<-(string linkModuleName) do {
    ...
} where:

   LinkRef is a variable of type Link or LinkRef
**Operation**

Assigns the variable `LinkRef` to be each successive incoming link arriving at object `tgtObject` via link module named `linkModuleName`. The string `linkModuleName` can be a specific link module name, or the string "*" meaning any link module.

Iterates through all incoming link references including those from baselines and soft-deleted modules.

**Note:** This loop only assigns to `LinkRef` incoming link values for which the source object is loaded; unloaded links are not detected.

**Example**

```dxl
LinkRef lr
for lr in current Object<"*" do {
    string name = fullName(source(lr))
    print name "\n"
}
```

**Syntax**

```dxl
for srcModuleName in Object tgtObject<-(string linkModuleName) do {
    ...
}
```

where:

- `srcModuleName` is a string variable
- `tgtObject` is a variable of type `Object`
- `linkModuleName` is a string variable

**Operation**

Assigns the variable `srcModuleName` to be the unqualified name of the source module of each successive incoming link arriving at object `tgtObject` via link module named `linkModuleName`. The string `linkModuleName` can be a specific link module name, or the string "*" meaning any link module.

**Note:** This loop assigns to `modName` values for all incoming links, whether the source is loaded or not. This can be used to pre-load all incoming link sources before using the for all incoming links loop.
Example

This example prints the unqualified name of all the source modules for incoming links to the current object:

```dxl
Object o = current
string srcModName
for srcModName in each (o<"*") do print srcModName "\n"
```

Syntax

```dxl
for srcModName in Object tgtObject<-(string linkModName) do {
  ...
}
```

where:

- `srcModName` is a string variable
- `tgtObject` is a variable of type Object
- `linkModName` is a string variable

Operation

Assigns the variable `srcModName` to be the unqualified name of the source module of each successive incoming link arriving at object `tgtObject` via link module named `linkModuleName`. The string `linkModuleName` can be a specific link module name, or the string "*" meaning any link module.

Includes links from baselines and soft-deleted modules, returning the name of the source module (without baseline version numbers).

Note: This loop assigns to `modName` values for all incoming links, whether the source is loaded or not. This can be used to pre-load all incoming link sources before using the for all incoming links loop.

Example

This example prints the unqualified name of all the source modules for incoming links to the current object:

```dxl
Object o = current
string srcModName
for srcModName in each (o<"*") do print srcModName "\n"
```
for all source references

Syntax

for srcModRef in Object tgtObject<-(string linkModName) do {
    ...
}

where:

srcModRef is a variable of type ModName_
tgtObject is a variable of type Object
linkModName is a string variable

Operation

Assigns the variable srcModRef to be the reference of the source module of each successive incoming link arriving at object tgtObject via link module named linkModuleName. The string linkModuleName can be a specific link module name, or the string "*" meaning any link module.

Note: This loop assigns to modName values for all incoming links, whether the source is loaded or not. This can be used to pre-load all incoming link sources before using the for all incoming links loop.

Example

Object o = current
ModName_ srcModRef
for srcModRef in (o<"*") do
    read(fullName(srcModRef), false)

for each source reference

Syntax

for srcModRef in Object tgtObject<-(string linkModName) do {
    ...
}

where:

srcModRef is a variable of type ModName_
tgtObject is a variable of type Object
linkModName is a string variable
**Operation**

Assigns the variable `srcModRef` to be the reference of the source module of each successive incoming link arriving at object `tgtObject` via link module named `linkModuleName`. The string `linkModuleName` can be a specific link module name, or the string "*" meaning any link module.

Includes links from baselines and soft-deleted modules.

**Note:** This loop assigns to `modName` values for all incoming links, whether the source is loaded or not. This can be used to pre-load all incoming link sources before using the for all incoming links loop.

**Example**

```dxl
Object o = current
ModName_ srcModRef
for srcModRef in (o<-"*") do
    read(fullName(srcModRef), false)
```

---

### for all link references

**Syntax**

```dxl
for linkRef in Object tgtObject<-(string linkModName) do {
    ...
}
```

**where:**

- `linkRef` is a variable of type `LinkRef`
- `tgtObject` is a variable of type `Object`
- `linkModName` is a string variable

**Operation**

Assigns the variable `linkRef` to be the link reference of each successive incoming link arriving at object `tgtObject` via link module named `linkModuleName`. The string `linkModuleName` can be a specific link module name, or the string "*" meaning any link module.

---

### for link module descriptor in folder

**Syntax**

```dxl
for linkModDesc in f do {
    ...
}
```
where:

- $linkModDesc$ is a variable of type `LinkModuleDescriptor`
- $f$ is a variable of type `Folder`

**Operation**
Assigns the variable $linkModDesc$ to be each successive link module descriptor in folder $f$.

---

**Versioned links**

---

**for all outgoing links**

**Declaration**

```dxl
for outLink in all Object srcObject-> (string linkModName) do {
    ...
}
```

where:

- $outLink$ is a variable of type `Link`
- $srcObject$ is a variable of type `Object`
- $linkModName$ is a variable of type `string`

**Operation**
This will iterate through all outgoing links, including links to baselines.

---

**for all incoming links**

**Declaration**

```dxl
for inLink in all Object tgtObject<- (string linkModuleName) do {
    ...
}
```

where:

- $inLink$ is a variable of type `Link` or `LinkRef`
- $tgtObject$ is a variable of type `Object`
Operation

These will iterate through all incoming links, including links from baselines.

for all source links

Declaration

for srcModName in Object tgtObject<-(string linkModName) do {
  ...
}

where:

srcModName  is a string variable

tgtObject is a variable of type Object

linkModName  is a string variable

This would include links from baselines, returning the name of the source module, without baseline version numbers.

for all source link references

Declaration

for srcModRef in Object tgtObject<-(string linkModName) do {
  ...
}

where:

srcModRef is a variable of type ModName_

tgtObject is a variable of type Object

linkModName  is a string variable

Operation

This would include links from baselines.
sourceVersion

Declaration

ModuleVersion sourceVersion(Link|LinkRef l)

Operation

This will return document version information for the source module of the specified Link or LinkRef. The new ModuleVersion type gives access to ModName_ and Baseline information, via new perms detailed in the rest of this section.

targetVersion

Declaration

ModuleVersion targetVersion(Link l)

Operation

This will return document version information for the target module of the specified Link.

echoed outlinks

Declaration

bool echo(Link l)

Operation

This returns true for an echoed outlink. An echoed outlink is any outgoing link in a module baseline which does not have a corresponding inlink in the target module leading back to this baseline. Any outgoing link in a baseline will be an echoed link unless it is a link to the same module or a link to another module in the same Baseline Set.

echoed inlinks

Declaration

bool echo(LinkRef l)

Operation

This returns true for an echoed inlink. An echoed inlink is any incoming link in a module baseline which does not have a corresponding outlink in the source module leading back to this baseline. Any incoming link in a baseline will be an echoed link unless it is a link from the same module or a link from another module in the same Baseline Set.
getSourceVersion(Linkset)

Declaration

ModuleVersion getSourceVersion(Linkset ls)

Operation

Returns some description of the version of the document in the source of a linkset ls.

Link management

This section defines functions for managing links. Links are referred to by the Link or LinkRef data type.

addLinkModuleDescriptor

Declaration

string addLinkModuleDescriptor(Folder f, 
  string source, 
  string target, 
  bool overrideable, 
  [bool mandatory, ] 
  string linkmod, 
  string desc)

Operation

Creates a new link module descriptor for the link between source and target, via link module linkmod, in folder f. If the link module does not exist when this function is called, desc is the description of the link module created. Folder f must be the folder that contains the module source.

If the operation succeeds, returns a null string; otherwise, returns an error message.

This function checks for duplicate source/target pairings. If the new link module descriptor would create a duplicate, it returns a message.

The overrideable parameter specifies whether the link module descriptor will be overrideable.

The optional mandatory parameter specifies whether the link module descriptor will be mandatory.

For further information on link module descriptors, see “About links and link module descriptors,” on page 393.
removeLinkModuleDescriptor

Declaration

```plaintext
string removeLinkModuleDescriptor(Folder f, string s, string t)
```

Operation

Deletes one link module descriptor defined for source `s` and target `t`, in folder `f`. If there is more than one `s/t` pair, the duplicates remain.

If the operation succeeds, returns a null string; otherwise, returns an error message.

For further information on link module descriptors, see “About links and link module descriptors,” on page 393.

setLinkModuleDescriptorsExclusive

Declaration

```plaintext
void setLinkModuleDescriptorsExclusive(Folder f, ModName_ m, bool flag)
```

Operation

Setting the boolean variable `flag` to `true` has the same effect as selecting the only allow outgoing links as specified in the above list option in the user interface.

Example

```plaintext
Folder f = current
ModName_ m = module("/A Project/A Module")
setLinkModuleDescriptorsExclusive(f, m, true)
```

getLinkModuleDescriptorsExclusive

Declaration

```plaintext
bool getLinkModuleDescriptorsExclusive(Folder f, ModName_)
```

Operation

Returns `true` if the only allow outgoing links as specified in the above list user interface option is set for the specified document. The specified document must be a child of the specified folder.

getDescription

Declaration

```plaintext
string getDescription(LinkModuleDescriptor linkModDesc)
```
Operation

Returns the description of the link module in the specified link module descriptor.
If the operation succeeds, returns a string; otherwise, returns null.
For further information on link module descriptors, see “About links and link module descriptors,” on page 393.

getName

Declaration

string getName(LinkModuleDescriptor linkModDesc)

Operation

Returns the name of the specified link module descriptor.
If the operation succeeds, returns a string; otherwise, returns null.
For further information on link module descriptors, see “About links and link module descriptors,” on page 393.

ggetSourceName

Declaration

string getSourceName(LinkModuleDescriptor linkModDesc)

Operation

Returns the name of the source in the specified link module descriptor.
If the operation succeeds, returns a string; otherwise, returns null.
For further information on link module descriptors, see “About links and link module descriptors,” on page 393.

gToUpper

Declaration

string getToUpper(LinkModuleDescriptor linkModDesc)

Operation

Returns the uppercase name of the specified link module descriptor.
If the operation succeeds, returns a string; otherwise, returns null.
For further information on link module descriptors, see “About links and link module descriptors,” on page 393.

gToLower

Declaration

string gToLower(LinkModuleDescriptor linkModDesc)

Operation

Returns the lowercase name of the specified link module descriptor.
If the operation succeeds, returns a string; otherwise, returns null.
For further information on link module descriptors, see “About links and link module descriptors,” on page 393.

gExtractName

Declaration

string gExtractName(LinkModuleDescriptor linkModDesc)

Operation

Returns the name of the specified link module descriptor.
If the operation succeeds, returns a string; otherwise, returns null.
For further information on link module descriptors, see “About links and link module descriptors,” on page 393.

getTargetName

Declaration

string getTargetName(LinkModuleDescriptor linkModDesc)

Operation

Returns the name of the target in the specified link module descriptor.
If the operation succeeds, returns a string; otherwise, returns null.
For further information on link module descriptors, see “About links and link module descriptors,” on page 393.
getOverridable

Declaration
bool getOverridable(LinkModuleDescriptor linkModDesc)

Operation
Returns whether the specified link module descriptor is overridable.
If the operation fails, returns null.
For further information on link module descriptors, see “About links and link module descriptors,” on page 393.

setOverridable

Declaration
void setOverridable(LinkModuleDescriptor linkModDesc, bool overridable)

Operation
If overridable is true, sets linkModDesc to overridable; otherwise sets linkModDesc to not overridable.
If linkModDesc is already overridable, the call fails. You can obtain the value of the override setting using the getOverridable function.
For further information on link module descriptors, see “About links and link module descriptors,” on page 393.

getMandatory

Declaration
bool getMandatory(LinkModuleDescriptor linkModDesc)

Operation
Returns whether the specified link module descriptor is mandatory.
If the operation fails, returns null.

setMandatory

Declaration
void setMandatory(LinkModuleDescriptor linkModDesc, bool mandatory)

Operation
If mandatory is true it sets linkModDesc to mandatory; otherwise sets linkModDesc to not mandatory.
If linkModDesc is already mandatory, the call fails.
delete(link)

**Declaration**

```c
void delete(Link l)
```

**Operation**

Marks link \( l \) for deletion. The delete only takes effect when the DXL script ends, or when the `flushDeletions` function is called.

module(link)

**Declaration**

```c
Module module(Link l)
```

**Operation**

Returns the link module handle of link \( l \), where linksets are stored as objects.

source

**Declaration**

```c
string source({Link|LinkRef} l)
ModName_ source({Link|LinkRef} l)
Object source(Link l)
```

**Operation**

The first form returns the unqualified name of the module that is the source of \( l \), which can be of type `Link` or `LinkRef`.

The second form returns a reference to the module that is the source of \( l \), which can be of type `Link` or `LinkRef`.

The third form returns the source object of link \( l \).

**Example**

```dxl
Object o = current
LinkRef lref
ModName_ srcModRef
for lref in o<-*"*" do {
    srcModRef = source lref
    read(fullName(srcModRef), true)
}
```
sourceAbsNo

Declaration
int sourceAbsNo({Link|LinkRef} l)

Operation
Returns the absolute number of the object that is the source of l, which can be of type Link or LinkRef.

target

Declaration
string target(Link l)
ModName_ target(Link l)
Object target(Link l)

Operation
The first form returns the unqualified name of the module that is the target of link l.
The second form returns a reference to the module that is the target of link l.
The third form returns the target object of link l. Returns null if the target module is not loaded, in which case your program can load the module and re-run target.

Example
Object o = current
Link lnk
ModName_ targetMod
for lnk in o->"*" do {
  targetMod = target lnk
  read(fullName(targetMod), true)
}

targetAbsNo

Declaration
int targetAbsNo(Link l)

Operation
Returns the absolute number of the object that is the target of l.
Default link module

This section defines functions that operate on the default link module, which is used by drag-and-drop operations from the Rational DOORS user interface.

get DefaultLinkModule

Declaration

string getDefaultLinkModule([ModName_ srcRef, ModName_ trgRef])

Operation

Returns the name of the default link module.

Example

print getDefaultLinkModule(module("Functional Requirements"),module("User Requirements"))

setDefaultLinkModule

Declaration

void setDefaultLinkModule(string linkModName)

Operation

Sets the name of the default link module.

Linksets

This section defines functions that apply to linksets. Linksets are referred to by the Linkset data type.

create(linkset)

Declaration

Linkset create([Module linkMod,] string source, string target)
Operation
Creates a linkset between modules specified by the strings source and target, in the link module linkMod. If linkMod is omitted, creates a linkset in the current module. If the link module is open for display, the display updates to show this linkset.

delete(linkset)

Declaration
void delete(Linkset ls)

Operation
Deletes the linkset ls. If the linkset is currently being displayed, the link module resets to displaying no linkset.

getSource getTarget

Declaration
string getSource(Linkset ls, Object &o)
string getTarget(Linkset ls, Object &o)

Operation
The first function gets the current source object in linkset ls, and sets object variable o to it.
The first function gets the current target object in linkset ls, and sets object variable o to it.
Either function returns null if it succeeds; otherwise, returns an error message.

linkset

Declaration
Linkset linkset(Object ls)

Operation
Converts a link module’s object ls into a linkset handle, which can be used with the operations load and delete.

Example
In this example, m must be a link module, which means that the objects it contains are linksets. To make this explicit the function linkset is called.
Module m = current
Object o
Linkset ls = linkset o
delete ls

load

**Declaration**

```dxl
definition load(Linkset ls)
```

**Operation**

Load the linkset `ls`. If the associated link module is open for display, the display updates to show this linkset.

setSource, setTarget

**Declaration**

```dxl
definition setSource(Linkset ls, Object o)
definition setTarget(Linkset ls, Object o)
```

**Operation**

Sets either the source or the target object in the linkset `ls`, as displayed in the link module window matrix view, to be object `o`. They depend on the module being visible.

If the operation succeeds, returns `null`; otherwise, returns an error message.

side1

**Declaration**

```dxl
definition side1(Module linkMod)
```

**Operation**

Returns the object that is currently selected on `side1` (the source side) of the linkset. Depends on the module being visible.

**Note:** When using this perm just after opening the module `linkMod`, the `refresh` perm should be used beforehand.

side2

**Declaration**

```dxl
definition side2(Module linkMod)
```

**Operation**

Returns the object that is currently selected on `side2` (the target side) of the linkset. Depends on the module being visible.
**Note:** When using this perm just after opening the module `linkMod`, the `refresh` perm should be used beforehand.

### unload

**Declaration**

```c
void unload(Linkset ls)
void unload(Module linkMod)
```

**Operation**

Unloads a loaded linkset specified by either the linkset handle `ls`, if it is current, or the current linkset of the link module `linkMod`.

---

### getTargetModule

**Declaration**

```c
ModName_ getTarget(Linkset ls)
```

**Operation**

Returns the target module reference for the specified linkset.

---

### External Links

**ExternalLink**

*ExternalLink* is a new data type representing the end of an external link. An external link is a one way link to the resource it references. No corresponding link is created in the linked resource.

**ExternalLinkDirection**

**Declaration**

```c
ExternalLinkDirection extLinkDir
```

**Operation**

Used to describe the direction of an external link. Valid values are `inward` and `outward`. 
ExternalLinkBehavior

**Declaration**

```
ExternalLinkBehaviour extLinkBeh
```

**Operation**

Used to describe the behavior of an external link. Valid values are `none` and `openAsURL`.

ExternalLink current

**Operation**

Fetches the current external link. This perm will return non-null only when called from within attribute DXL executing against an external link. In all other cases it will be null.

**Example**

```
External extLink = current
```

create(external link)

**Declaration**

```
string create(Object o,
            string description,
            string name,
            ExternalLinkDirection extLinkDir,
            ExternalLinkBehaviour extLinkBeh,
            string body,
            ExternalLink& extLink)
```

**Operation**

Creates an external link on the specified object. The object must be locked and be modifiable by the current session. On success, null is returned and the new link is returned in the `ExternalLink&` variable.

canDelete(external link)

**Declaration**

```
bool canDelete(ExternalLink extLink)
string canDelete(ExternalLink extLink)
```

**Operation**

This perm should always return `false`. If applied to a link from a baseline, an error string will be returned.
source

Declaration

Object source(ExternalLink extLink)

Operation

Returns information concerning the object having this external link for external links marked as out. For external links marked as in, the perm returns null.

for all outgoing external links

Declaration

for extLink in (Object o) -> string ""

where:

extLink is a variable of type ExternalLink

Operation

Iterates over all external outgoing links on the object o. The supplied string parameter must be the empty string.

for all incoming external links

Declaration

for extLink in (Object o) <- string ""

where:

extLink is a variable of type ExternalLink

Operation

Iterates over all external incoming links on the object o. The supplied string parameter must be the empty string.

Example

The following example demonstrates the external link behavior. It must be executed from within a module that has at least one object.

ExternalLink el, el1, el2, el3

//Create 3 external links

print create(current Object, "Description1", "Name1", outward, none, "https://www.ibm.com", el1)

print create(current Object, "Description2", "Name2", outward, openAsURL, "https://www.ibm.com/software/support/", el2)
print create(current Object, "Description3", "Name3", inward, openAsURL, "https://www.ibm.com/software/support", el3)

follow(el1) //This will fail - follow behavior is 'none'. update("IBM Web Site", name(el1), direction(el1), openAsURL, body(el1), el1) follow(el1)

Object o = current
//Iterate over outward links
for el in o->"" do
{
    print "Created on " el."Created On" " Last modified on " el."Last Modified On" "\n"
}

//Iterate over inward links - changing External Link data
for el in o<-"" do
{
    string elName = name(el)
    string elDesc = description(el)
    ExternalLinkDirection elDir = direction(el)

    ExternalLinkBehaviour elBehaviour= behaviour(el)
    string elBody = body(el)

    if (elBehaviour == none)
    {
        elName = "New name"
    }
    update(elDesc, elName, elDir, elBehaviour, elBody, el)
}

for el in o<-"" do
{
    string elName = name(el)
    print "" elName "" created on " el."Created On" " Last modified on " el."Last Modified On" "\n"
if (behaviour(el) == openAsURL)
{
    print "Opening up " body(el) "\n"
    print follow(el) "\n"
    update("IBM Support Web Site", name(el), direction(el), behaviour(el), body(el), el)
    break
}

for all incoming and outgoing external links

**Declaration**

for extLink in (Object o) <-> string ""
where:

*extLink* is a variable of type *ExternalLink*

**Operation**

Iterates over all external links on the object *o*, both incoming and outgoing. The supplied string parameter must be the empty string.

**Example**

To print the URLs of all Validated By links in the currently selected object.

Object *o* = current
ExternalLink el
string url
for el in all o<->"" do
{
    url = body el
    print url "\n"
}
OSLC Link Discovery

When OSLC (external) links are discovered the results are stored in DOORS in a database-wide cache so that future sessions that open modules with those links open faster. When a user opens a module, the cache is checked first for any external links. If the data in the cache has not yet expired then the cached external links are shown; else, a new query is executed to discover any OSLC (external) links and the cache is then updated with the results. The cache has a default expiry time of 5 minutes after which the external links are considered to be out of date. This expiry time can be modified.

**getCachedExternalLinkLifeTime**

**Declaration**

```plaintext
int getCachedExternalLinkLifeTime()
```

**Operation**

Returns the life time (expiry time) of the cached external links in seconds.

**setCachedExternalLinkLifeTime**

**Declaration**

```plaintext
string setCachedExternalLinkLifeTime(int lifetime)
```

**Operation**

Sets the life time (expiry time) of the cached external links to lifetime seconds.

If the value lifetime is zero then this will disable link discovery.

Returns an error if the user does not have the manage database privilege; otherwise, returns null.

**discoverLinks**

**Declaration**

```plaintext
string discoverLinks({Object obj | Module m} [, bool reset])
```

**Operation**

Runs link discovery queries for the specified object or all objects and views of the specified module to find out possible links.

If 'reset' is true, then existing errors and timestamps are cleared before starting link discovery. This essentially forces a fresh link discovery.

Returns error message when it fails, returns NULL otherwise.
**linksDiscovered**

**Declaration**

```c
bool linksDiscovered({Object obj | Module m})
```

**Operation**

Returns true if link discovery is completed for the specified object or all objects and views of the specified module.

**discoverLinksForViews**

**Declaration**

```c
string discoverLinksForViews(Module m [, bool reset])
```

**Operation**

Runs link discovery for all views of the specified module.

If 'reset' is true, then existing errors and timestamps are cleared before starting link discovery. This essentially forces a fresh link discovery.

Returns error message when it fails, returns NULL otherwise.

**linksDiscoveredForViews**

**Declaration**

```c
bool linksDiscoveredForViews(Module m)
```

**Operation**

Returns true if link discovery is completed for all views of the specified module.

**discoverLinksForViewsAsync**

**Declaration**

```c
void discoverLinksForViewsAsync(Module m [, bool reset])
```

**Operation**

Runs link discovery for all views of the given module. This is an asynchronous operation so does not block.

If 'reset' is true, then existing errors and timestamps are cleared before starting link discovery. This essentially forces a fresh link discovery.
discoverLinksAsync

Declaration

void discoverLinksAsync(Module m [, bool reset])

Operation

Runs link discovery queries for all objects and views of the module to find out possible links. This is an asynchronous operation so does not block.

If 'reset' is true, then existing errors and timestamps are cleared before starting link discovery. This essentially forces a fresh link discovery.

Rational DOORS URLs

getURL and getResourceURL

Declaration

string getResourceURL(Database__ d)
string getResourceURL(Module m)
string getResourceURL(ModName_ modName)
string getResourceURL(ModuleVersion modVer)
string getResourceURL(Object o)
string getResourceURL(Folder f)
string getResourceURL(Project p)
string getResourceURL(Item i)

Note: You can replace getResourceURL with getURL perm.

Operation

The getResourceURL perm returns the same value as the Rational DOORS URL displayed in the properties window.
The getURL perm is only available for compatibility reasons and does not return the dwaHost, dwaPort, and dwaProtocol settings.

decodeURL

Declaration

string decodeURL(string url, string& dbHost, int& dbPort, string& dbName, string& dbId, Item& i, ModuleVersion& modVer, int& objectAbsno)
Operation

This perm decodes the given Rational DOORS URL and returns in its output parameters enough details to validate the URL `url` against the current database and navigate to the item or module specified by that URL.

The output Item `i` and ModuleVersion `modVer` will be null if the URL refers to the database root node.

The output ModuleVersion will be null if the URL refers to a project or folder.

The `objectAbsno` variable will be -1 unless the URL specifies navigation to a particular object.

The function returns null if the URL is successfully decoded, or an error string if the referenced Item cannot be found or the user does not have read access to the referenced Item.

This perm only works on legacy Rational DOORS URLs. This perm does not work when the re-director is enabled for Rational DOORS, for example when the URLs have been transformed using the `-urlPrefix` switch in `dbadmin`.

In this case, convert the URLs to legacy URLs using the perm `getLegacyURL()`.

Example

The following example demonstrates the Rational DOORS URL behavior. The current example returns the details for the current Object selected in a module. The second last line of the example can be changed to return details for the corresponding item.

```dxl
string urlInfo(string url)

// DESCRIPTION: Returns a string describing the target of the specified URL string.
{
    string result = null
    ModuleVersion mv
    int objectAbsno
    Item i

    string dbHost = null
    int dbPort
    string dbName
    string dbID = null

    result = decodeURL(url, dbHost, dbPort, dbName, dbID, i, mv, objectAbsno)

    if (null result)
    {
        if (dbID != getDatabaseIdentifier)
        {
```
result = "The dbID does not match the current database."
}
else if (null i)
{
    result = "Database: " dbName ""
}
else if (null mv)
{
    // we're going to the top level node
    result = (type i) ": " (fullName i) ": " (description i)
}
else
{
    // it's a module or baseline
    Module m = null

    if (isBaseline(mv))
    {
        result = "Baseline: " (fullName mv) " [" (versionString mv) "]: " (description module mv)
    }
    else
    {
        result = "Module: " (fullName mv) ": " (description module mv)
    }

    if (objectAbsno >= 0)
    {
        if (isBaseline(mv))
        {
            m = load(mv, true)
        }
        else
        {
        }
string mode = getenv("DOORSDEFOPENMODE")
if (mode == "READ_ONLY" || mode == "r")
{
    m = read(fullName(mv))
}
else if (mode == "READ_WRITE_SHARED" || mode == "s")
{
    m = share(fullName(mv))
}
else
{
    // Check the rights for the user and open the module as per the rights
    if (canModify(i))
    {
        m = edit(fullName(mv))
    }
    else
    {
        m = read(fullName(mv))
    }
}
}
if (null m)
{
    // Something went wrong
    result = result "\nCould not open module " (fullName mv) "." 
}
else
{
    current = m
    Object o = gotoObject(objectAbsno, m, true)
    if (null o)
    {

result = result "\nCould not locate object " objectAbsno "." 
}
else {
    result = result "\nObject " objectAbsno ""
}
if (!null o."Object Heading" && length(o."Object Heading" "") > 0) {
    result = result "\nObject Heading: " o."Object Heading" ""
}
if (!null o."Object Text" && length(o."Object Text" "") > 0) {
    result = result "\nObject Text: " o."Object Text" ""
}
}
return result
}

string obj_url = getURL(current Object)
print urlInfo(obj_url)

getlegacyURL

Declaration
string getLegacyURL(object o)

Operation
This perm returns the legacy Rational DOORS URL. The legacy URL contains the protocol as "doors". This URL can then be decoded using decodeURL.

Example
ModuleVersion mv
int objectAbsno
Item i
string dbHost = null
int dbPort
string dbName
string dbID = null

string objUrl = getURL(current Object)

string legacyUrl
string errorMsg
errorMsg = getLegacyURL(objUrl, legacyUrl)
if(!null errorMsg)
{
    print errorMsg "\n"
}
else
{
    errorMsg = decodeURL(legacyUrl, dbHost, dbPort, dbName, dbID, i, mv, objectAbsno)
}
if(!null errorMsg)
{
    print errorMsg "\n"
}
else
{
    print "Original URL - " objUrl "\nDB Host - " dbHost "\n"
    print "DB Port - " dbPort "\nDB Name - " dbName "\nDB Id - " dbId "\nAbsolute Number - " objectAbsno "\n"
}

validateDOORSURL

Declaration

string validateDOORSURL(string url)
Operation

This perm takes a Rational DOORS URL and performs a basic check that the URL structure is correct and required elements are present.

The function returns NULL if the URL is successfully validated, or an error string if there is a problem.

Example

Object o = current
string url = getURL o
string s = validateDOORSURL(url)

if (null s){
    print "URL is valid"
} else {
    print "Error in URL : " s
}

isDefaultURL

Declaration

bool isDefaultURL(string URL)

Operation

Returns true if the supplied URL does not have an explicitly specified protocol.

Example

string url = "www.google.com"
string fullURL

if (isDefaultURL(url)){
    fullURL = "http://" url
}
print fullURL

getResourceURL

Declaration

string getResourceURL(Module|Object|Database__|ModuleVersion|ModName__|Folder|Project|Item)
Operation

Returns the resource URL of the passed in item.

---

**getResourceURLConfigOptions**

Declaration

```c
void getResourceURLConfigOptions(string &dwaProtocol, string &dwaHost, int &dwaPort)
```

Operation

Gets the `dwaProtocol`, `dwaHost`, and `dwaPort` DBAdmin options configured for this database. The `dwaProtocol`, `dwaHost`, and `dwaPort` parameters contain the values upon return.

---

**decodeResourceURL**

Declaration

```c
string decodeResourceURL(string resourceURL, string &protocol, string& dbHost, int& dbPort, string& repositoryId, string& dbName, string& dbId, Item&, ModuleVersion&, string& viewName, int& objectAbsno)
```

Operation

Breaks down a passed-in resource URL into its constituent parts and passes back the information as may be applicable into the reference parameters.

Returns `null` on success, error message on failure.
Chapter 20
Attributes

This chapter describes the use of Rational DOORS attributes from DXL:
• Attribute values
• Attribute value access controls
• Multi-value enumerated attributes
• Attribute definitions
• Attribute definition access controls
• Attribute types
• Attribute type access controls
• Attribute type manipulation
• DXL attribute

Attribute values

This section defines constants, operators and functions for working with attribute values. Attribute values are one of the most important aspects of Rational DOORS.
Many example DXL programs in this manual or in the DXL library use attribute values.

maximumAttributeLength

Declaration

int maximumAttributeLength

Operation

Defines a constant, which equates to the maximum number of characters in a string attribute.

Attribute value extraction

Attribute names are available for use in combination with the . (dot) operator to extract the value of attributes. The syntax for using the attribute names is:

(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)
(ModuleProperties mp).(string attrName)
where:

\text{o} \quad \text{is an object of type Object}
\text{m} \quad \text{is a variable of type Module}
\text{l} \quad \text{is a variable of type Link}
\text{mp} \quad \text{is a variable of type ModuleProperties}
\text{attrName} \quad \text{is a string identifying the attribute}

This means that you can write:

\text{o."Object Heading"}
\text{m."Description"}
\text{l."Created By"}

when you want to refer to the values of a named attribute of object \text{o}, module \text{m} or link \text{l}.

A selected attribute can be assigned the value of a DXL variable (see “Assignment (to attribute),” on page 431). Conversely, a DXL variable can be assigned the value of an attribute (see “Assignment (from attribute),” on page 430).

\textbf{Concatenation (attribute)}

The space character is the concatenation operator, which is shown as <space> in the following syntax:

\text{attrRef <space> string s}

Concatenates string \text{s} onto \text{attrRef} and returns the result as a string.

Unlike assignment, the attribute can be of any type, because Rational DOORS automatically converts the value to a string.

\textbf{Example}

\text{string s = "Created On " (current Object)."Created On" \"\n"}

\textbf{Assignment (from attribute)}

The assignment operator \text{=} can be used as shown in the following syntax:

\text{bool b = attrRef}
\text{int i = attrRef}
\text{real r = attrRef}
\text{string s = attrRef}
\text{Date d = attrRef}
where `attrRef` is in one of the following formats:

(Object `o`).(string `attrName`)
(Module `m`).(string `attrName`)
(Link `l`).(string `attrName`)

**Operation**

These assign the value of the referenced attribute `attrRef` to bool `b`, int `i`, real `r`, string `s`, or Date `d`. Boolean assignment is slightly unusual in that it enables the retrieval of the value of an enumeration attribute with two elements, such as an attribute of type Boolean. The first element in the enumeration maps to false; the second element maps to true.

All assignments return the result of the assignment.

**Example**

Object `o` = current
Module `m` = current
Link `l`

int `i` = o."Absolute Number"
real `r`
if (exists attribute "Cost")
    `r` = o."Cost"
else
    `r` = 0.0
string `s` = o."Created By"
Date `d` = o."Created On"
bool `b` = o."OLE"
print `i" "r" "s" "d" "b"
for `l` in o->"*" do {
    string `s1` = `l"Last Modified By"
    print `s1"\n"
}
string desc = m."Description"
print desc "\n"
int `i2`, `i3`
i3 = i2 = o."Absolute Number"

---

**Assignment (to attribute)**

The assignment operator `=` can be used as shown in the following syntax:

`attrRef = bool b`
attrRef = int i
attrRef = real r
attrRef = string s
attrRef = Buffer b
attrRef = Date d

where attrRef is in one of the following formats:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation
Assigns bool b, int i, real r, string s, Buffer b, or Date d to the attribute reference attrRef.
Again, boolean assignment enables the setting of an enumeration attribute that has two elements in its definition, such as an attribute of type Boolean.

Example
Object o = current
o."Object Heading" = "Front Matter"
o."Integer Attribute" = 2
o."Accepted" = false

canRead, canWrite(attribute)

Declaration
bool canRead(Module m,
    string attrName)
bool canWrite(Module m,
    string attrName)
bool canRead(attrRef)
bool canWrite(attrRef)

where attrRef is in one of the following formats:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation
The first two forms return whether the current Rational DOORS user can read or write values of the attribute name attrName in module m.
The third and fourth forms allow you to use the dot notation directly.

Example

// Test current user permission
Module m
const string ACreatedBy = "Created By"
if (!canWrite(m, ACreatedBy) &&
   canRead(m, ACreatedBy)) {
    print "I can only read.\n"
}
// Use dot notation
Object o = current
const string ACreatedBy = "Created By"
if (!canWrite o.ACreatedBy && canRead o.ACreatedBy) {
    print "I can read the attribute but I cannot
        write to it.\n"
}

type(attribute)

Declaration

string type(Module m,
           string attrName)
string type(attrRef)
where attrRef is in one of the following formats:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation

The first form returns the name of the type of the attribute named \texttt{attrName} in module \texttt{m}.
The second form enables you to use the dot notation directly.

Example

// Use dot notation
print (type (current Object)."Object Heading") "\n"
// Use module
print (type (current Module,"Object Heading")) "\n"
for module attributes in module

**Syntax**

```dxl
for attribute in attributes(module) do {
    ...
}
```

where:

- `attribute` is a string variable
- `module` is a variable of type `Module`

**Operation**

Assigns the string `attribute` to be each successive attribute that is defined for `module`.

**Example**

```dxl
string modAttrName
for modAttrName in attributes (current Module) do
    print modAttrName "\n"
```

for object attributes in module

**Syntax**

```dxl
for objAttrName in module do {
    ...
}
```

where:

- `objAttrName` is a string variable
- `module` is a variable of type `Module`

**Operation**

Assigns the string `objAttrName` to be each successive attribute that is defined for objects in `module`.

**Example**

```dxl
string objAttrName
for objAttrName in (current Module) do print objAttrName "\n"
```
unicodeString

**Declaration**

```plaintext
string unicodeString(attrRef)
```

where `attrRef` is in one of the following formats:

- `(Object o).(string attrName)`
- `(Module m).(string attrName)`
- `(Link l).(string attrName)`

**Operation**

Returns the value of the specified attribute as plain text. If the attribute contains rich text including characters in Symbol font, then these characters are converted to the Unicode equivalents.

**Example**

```plaintext
Object o = current
string s = unicodeString (o."Object Text")
print s "\n"
```

generatedUnicode

**Declaration**

```plaintext
string getBoundedUnicode(attrRef, int maxSize)
```

where `attrRef` is in one of the following formats:

- `(Object o).(string attrName)`
- `(Module m).(string attrName)`
- `(Link l).(string attrName)`

**Operation**

Returns a plain text value derived as in `unicodeString(attrRef)`, but limited to a maximum number of characters as specified by the `maxSize` argument.

**Example**

```plaintext
Object o = current
string s = getBoundedUnicode(o."Object Text", 11)
print s "\n"
```
Attribute value access controls

This section describes functions that report on access rights for an attribute value.

canCreate(attribute)

Declaration

bool canCreate(Module m,
   string attrName)

bool canCreate(attrRef)

where attrRef is in one of the following formats:

(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation

The first form returns true if the current Rational DOORS user can create values of the attribute that is named attrName in module m. Otherwise, returns false.

The second form enables you to use the dot notation directly.

canControl(attribute)

Declaration

bool canControl(Module m,
   string attrName)

bool canControl(attrRef)

where attrRef is in one of the following formats:

(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation

The first form returns true if the current Rational DOORS user can change the access controls on the attribute that is named attrName in module m. Otherwise, returns false.

The second form enables you to use the dot notation directly.
canModify(attribute)

Declaration

```dxl
bool canModify(Module m,
               string attrName)
```

where `attrRef` is in one of the following formats:

- `(Object o).(string attrName)`
- `(Module m).(string attrName)`
- `(Link l).(string attrName)`

Operation

The first form returns `true` if the current Rational DOORS user can modify values of the attribute that is named `attrName` in module `m`. Otherwise, returns `false`.

The second form enables you to use the dot notation directly.

canDelete(attribute)

Declaration

```dxl
bool canDelete(Module m,
               string attrName)
```

where `attrRef` is in one of the following formats:

- `(Object o).(string attrName)`
- `(Module m).(string attrName)`
- `(Link l).(string attrName)`

Operation

The first form returns `true` if the current Rational DOORS user can delete values of the attribute that is named `attrName` in module `m`. Otherwise, returns `false`.

The second form enables you to use the dot notation directly.

Multi-value enumerated attributes

This section defines functions that apply to multi-value enumerated attributes.
Assignment (enumerated option)

The assignment operators \( += \) and \( -= \) can be used as shown in the following syntax:

\[
\text{attrRef} \ += \text{string } s \\
\text{attrRef} \ -= \text{string } s
\]

where \text{attrRef} is in one of the following formats:

\[
\text{(Object } o\text{).}(\text{string } attrName) \\
\text{(Module } m\text{).}(\text{string } attrName) \\
\text{(Link } l\text{).}(\text{string } attrName)
\]

Operation

Adds or removes an enumerated option from the value of the attribute.

Example

This example adds "Australia" to the list of values of the attribute "Country" of the current object, and removes "Borneo".

Object \( o \) = current

\( o\text{."Country" } += \text{"Australia"} \)

\( o\text{."Country" } -= \text{"Borneo"} \)

isMember

Declaration

\[
\text{bool isMember(} attrRef, \\
\quad \text{string } s) \\
\text{where } attrRef \text{ is in one of the following formats:} \\
\text{(Object } o\text{).}(\text{string } attrName) \\
\text{(Module } m\text{).}(\text{string } attrName) \\
\text{(Link } l\text{).}(\text{string } attrName)
\]

Operation

Returns \text{true} if the option specified as \( s \) is present in the multi-value attribute.

Example

\[
\text{if (isMember((current Object)\text{."Country"}, \text{"Australia"}))} \{ \\
\quad \text{addRequirement("Right-hand drive model needed")} \\
\}
\]
Attribute definitions

This section defines functions and a for loop that manipulate Rational DOORS attribute definitions. The fundamental type that is used is AttrDef, which is a handle on an attribute definition.

**Note:** Reloading a module (for example, changing edit modes) in a DXL script removes any attribute definition values currently assigned to variables in that script. If a module is reloaded, reassign the attribute definitions.

Attribute definition properties

Properties are defined for use with the . (dot) operator and an attribute definition handle to extract information from an attribute definition, as shown in the following syntax:

```
(AttrDef ad).property
```

The following tables list the properties and the information they extract:

### String property | Extracts
---|---
`dxl` | DXL text of an attribute that uses DXL attribute.
`name` | The name of an attribute definition.
`typeName` | The name of the type of an attribute definition.
`description` | The description of the attribute definition.
`uri` | The URI of an attribute definition.

### Boolean property | Extracts
---|---
`canWrite` | Whether the user can delete the attribute definition.
`defval` | Whether the attribute definition is for an attribute that has a default value.
`dxl` | Whether the attribute definition is for an attribute that has its value generated by DXL.
`hidden` | Whether the attribute definition is for an attribute that is hidden. This function is provided only for forward compatibility with future releases of Rational DOORS.
`inherit` | Whether the attribute definition is for an attribute that is inherited.
`module` | Whether the attribute definition is defined for the module.
### Example

This example uses string properties:

```dxl
// name
AttrDef ad = find(current Module, "Object Text")
print ad.name // prints Object Text
// typeName
AttrDef ad = find(current Module, "Created On")
print ad.typeName // prints Date
```
// dxl
AttrDef ad = find(current Module, "DXL initialized attribute")
if (ad.dxl) {
    string dxlVal = ad.dxl
    print dxlVal "\n"
}

// useraccess
AttrDef ad
Module m = current
for ad in m do {
    print ad.name "-" ad.useraccess "\n"
}

This example uses boolean properties:
// object
AttrDef ad = find(current Module, "Description")
print ad.object          // prints false
// module
AttrDef ad = find(current Module, "Description")
print ad.module          // prints true
// system
if (thisAttr.system) {
    ack "System attribute: cannot delete"
}

// canWrite
AttrDef ad
Module m = current
for ad in m do{
    print ad.name "-" ad.canWrite "\n"
}

This example uses the property type:
AttrDef ad = find(current Module, "Description")
AttrType at = ad.type
print at.name          // prints String

This example uses the property defval for a default value of type string:
AttrDef ad = find(current Module, "Created Thru")
string def = ad.defval
print def              // prints Manual Input
Concatenation (attribute definition)

The space character is the concatenation operator. All the individual elements of an attribute definition can be concatenated.

create(attribute definition)

Syntax

```
AttrDef create([module|object]
    [property value]...
    [(default defVal)]
    attribute(string attrName))
```

Operation

Creates a new attribute definition called `attrName` from the call to `attribute`, which is the only argument that must be passed to `create`. The optional arguments modify `create`, by specifying the value of attribute properties. The arguments can be concatenated together to form valid attribute creation statements.

The keywords `module` and `object` specify that the attribute definition that is being created applies to modules or objects, respectively.

The default property specifies the default value for the attribute definition that is being created as `defVal`. This property should always be specified within parenthesis to avoid parsing problems. The value must be given as a string, even if the underlying type is different. Rational DOORS converts the value automatically.

As required, you can specify other properties. The defaults are the same as the Rational DOORS user interface.

<table>
<thead>
<tr>
<th>String property</th>
<th>Specifies</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dxl</code></td>
<td>The code that is associated with an attribute in <code>dxlcode</code>.</td>
</tr>
<tr>
<td><code>type</code></td>
<td>The type of the attribute definition as <code>typeName</code>.</td>
</tr>
<tr>
<td><code>description</code></td>
<td>The description of the attribute definition.</td>
</tr>
<tr>
<td><code>uri</code></td>
<td>The URI of an attribute definition.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boolean property</th>
<th>Specifies</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>changeBars</code></td>
<td>Whether the attribute definition that is being created alters change bars.</td>
</tr>
<tr>
<td><code>date</code></td>
<td>Whether the attribute definition that is being created alters dates.</td>
</tr>
<tr>
<td><code>hidden</code></td>
<td>Whether the attribute definition that is being created is hidden. Note that this function is only provided for forward compatibility with future releases of Rational DOORS.</td>
</tr>
</tbody>
</table>
Example

• This example builds an attribute named "Count" which has a default value of 0:
  ```dxl```
  create (default "0") attribute "Count"
  ```

• This example builds an integer attribute named "Cost" which applies to the module:
  ```dxl```
  create module type "Integer" attribute "Total Cost"
  ```

• This example builds an integer attribute named "Cost" which applies to the objects in the module, but not the module itself:
  ```dxl```
  create object type "Integer" attribute "Cost"
  ```

• This example uses some of the other attribute definition functions:
  ```dxl```
  create module type "String" (default "Help")
  history true //-
  changeBars false
  date false
  inherit true           //-
  hidden false
  attribute "Usage"
  ```

• This example creates an "Integer" attribute definition called "Cost2", which applies only to objects:
  ```dxl```
  create attribute "Cost2"
  ```

• This example creates a multi-valued attribute definition "attribute name", which uses the enumeration type "enumeration name" and sets its default to two values: value1 and value2.
  ```dxl```
  create type "enumeration name" (default "value1\nvalue2")
  multi true
  attribute "attribute name"
  ```
  A newline character must be used to separate the different values.

• This example defines code associated with attribute called "cost":
  ```dxl```
  AttrDef ad = create object type "Integer" attribute "cost"
  dxl "int i = 10 \n obj.attrDXLName = i "
  ```

delete(attribute definition)

Declaration

```dxl```
string delete([Module m,]
            AttrDef ad)
```dxl```

Operation

Deletes the attribute definition `ad` from module `m`. If `m` is omitted, deletes `ad` from the current module.
Example

```dxl
void deleteAttrDef(string s)
{
    string err
    AttrDef ad = find(current Module, s)
    err = delete(ad)
    if (err !="") ack err
}

deleteAttrDef "attribute_name"
```

exists

**Declaration**

```dxl
bool exists(attribute(string attributeName))
```

**Operation**

Returns true if the attribute named `attributeName` exists in the current module.

**Example**

```dxl
if (exists attribute "Cost")
    print "Cost is already there.\n"
```

find(attribute definition)

**Declaration**

```dxl
AttrDef find(Module m, 
    string attributeName)
```

**Operation**

Returns the attribute definition for the attribute named `attributeName` in module `m`.

**Example**

```dxl
AttrDef ad = find(current Module, "Object Heading")
```

attributeValue

**Declaration**

```dxl
bool attributeValue(AttrDef attrDef, 
    string s)
```
isAttributeValueInRange

Declaration

bool isAttributeValueInRange(AttrDef ad, attrRef)

where attrRef can be one of:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation

If the value of the attribute attrRef is within the range defined for Attribute Definition ad, then return true. Otherwise, return false.

Note: For attributes based on types that are not ranged, always returns true.

getBoundedAttr

Declaration

string getBoundedAttr(attrRef attrdef, int number)

where attrRef is in one of the following formats:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation

Returns the first number of characters of the value of attribute definition attrDef.

This is particularly useful when working with attribute values that could potentially be extremely large (for example, encoded picture data) but the entire value is not required.

Example

Object o = current
print getBoundedAttr(o."Object Text", 3)
hasSpecificValue

Declaration

```c
bool hasSpecificValue({Link l|Module m|Object o},
    AttrDef attrDef)
```

Operation

Returns `true` if the attribute definition `attrDef` has a specific value for link `l`, open module `m`, or object `o`. Otherwise, returns `false`.

isVisibleAttribute

Declaration

```c
bool isVisibleAttribute(AttrDef attrDef)
```

Operation

Returns `true` if the specified attribute is not a hidden attribute. Returns `false` if the specified attribute is a hidden attribute.

*Note:* This only applies to object attributes. It return `false` when used with module attributes.

modify(attribute definition)

Declaration

```c
AttrDef modify(AttrDef old,
    [setproperty value,]
    AttrDef new)
```

Operation

Modifies an existing attribute definition by passing it a new attribute definition. The optional second argument enables you to set a single property, as follows:

<table>
<thead>
<tr>
<th>String property</th>
<th>Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setDefault</code></td>
<td>The default string.</td>
</tr>
<tr>
<td><code>setDXL</code></td>
<td>The attribute to DXL code contained in the string argument.</td>
</tr>
<tr>
<td><code>setName</code></td>
<td>The attribute’s name in a string.</td>
</tr>
<tr>
<td><code>setDescription</code></td>
<td>The attribute description.</td>
</tr>
<tr>
<td><code>uri</code></td>
<td>The URI of an attribute definition.</td>
</tr>
</tbody>
</table>
Example 1

AttrDef ad = create object type "Integer" attribute "cost"
ad = modify(ad, object type "Integer" attribute "Costing")
ad = modify(ad, setHistory, true)
ad = modify(ad, setDefault, "123")
ad = modify(ad, setURI, "http://www.webaddress.com")

Example 2

This example uses Locale properties
Locale loc = userLocale
AttrDef ad = find(current Module, "Object Text")
Modify (ad, setLocale, loc)

for attribute definition in module

Syntax

for ad in {module|modProps} do {
    ...
}
where:

- \textit{ad} is a variable of type \textit{AttrDef}
- \textit{module} is a variable of type \textit{Module}
- \textit{modProps} is a variable of type \textit{ModuleProperties}

\textbf{Operation}

Assigns the attribute definition \textit{ad} to be each successive definition present in the module \textit{module}, or \textit{modProps}, provided the definition applies to either modules or objects.

\textbf{Example}

\begin{verbatim}
AttrDef ad
for ad in current Module do {
    print "Attribute: " ad.name "\n"
}
\end{verbatim}

\textbf{Syntax}

\begin{verbatim}
for ad in attributes \{mod|modprops\} do {
    ...
}
\end{verbatim}

where:

- \textit{ad} is a variable of type \textit{AttrDef}
- \textit{mod} is a variable of type \textit{Module}
- \textit{modprops} is a variable of type \textit{ModuleProperties}

\textbf{Operation}

Assigns \textit{ad} to be the name of each successive module level attribute definition in the supplied \textit{Module}, or \textit{ModuleProperties}.

\textbf{Attribute definition example program}

\begin{verbatim}
// attribute definition DXL example
\end{verbatim}
/*
   Example of Attribute Definition DXL
*/

void print(AttrDef ad) {   // print out some information on ad
    print ad.name " : "
    print "type \"" ad.typeName "\"
    // does ad apply to objects?
    print (ad.object ? " object " : "")
    // does ad apply to modules?
    print (ad.module ? " module" : "")
    print (ad.inherit ? " inherit" : "")
    // are values inherited?
    AttrType typ = ad.type
    if (typ.name == "Integer" && ad.defval) {
        // print any default int value
        int d = ad.defval
        print " default " d ""
    }
    if (ad.dxl) {
        string dxlVal = ad.dxl
        print " isDxl "" dxlVal """'
        // does ad use DXL attribute?
    }
    print "\n"
} // print

// main program
// create two new attributes
create object type "Integer" attribute "Cost"
create module type "Integer" attribute "Total Cost"
AttrDef ad
// print module attribute definitions
print "Module attribute definitions:\n"
for ad in current Module do
    if (ad.module)
        print ad
// print object attribute definitions
print "Object attribute definitions:\n"
for ad in current Module do
    if (ad.object)
        print ad

For a larger example of the use of AttrType, AttrDef and Rational DOORS attributes, see
$DOORSHOME/lib/dxl/utils/copyops.inc, which enables the copying of an attribute of an object in one
module to an object in another module. If the target module does not have the necessary attribute types and definitions,they are automatically constructed.

### Attribute definition access controls

This section describes functions that report on access rights for an attribute definition.

#### canCreateDef

**Declaration**

```c
bool canCreateDef(AttrDef attrDef)
```

**Operation**

Returns `true` if the current Rational DOORS user has create access to the attribute definition `attrDef`.

#### canCreateVal

**Declaration**

```c
bool canCreateVal(AttrDef attrDef)
```

**Operation**

Returns `true` if the current Rational DOORS user has create access to the value of the attribute definition `attrDef`.

#### canControlDef

**Declaration**

```c
bool canControlDef(AttrDef attrDef)
```

**Operation**

Returns `true` if the current Rational DOORS user can change the access controls on the attribute definition `attrDef`. 
canControlVal

**Declaration**

```c
bool canControlVal(AttrDef attrDef)
```

**Operation**

Returns true if the current Rational DOORS user can change the access controls on the value of the attribute definition `attrDef`.

---

canDeleteDef

**Declaration**

```c
bool canDeleteDef(AttrDef attrDef)
```

**Operation**

Returns true if the current Rational DOORS user can delete the attribute definition `attrDef`. Otherwise, returns false.

---

canDeleteVal

**Declaration**

```c
bool canDeleteVal(AttrDef attrDef)
```

**Operation**

Returns true if the current Rational DOORS user can delete the value of the attribute definition `attrDef`. Otherwise, returns false.

---

canCreateAttrDefs

**Declaration**

```c
bool canCreateAttrDefs(Module m)
```

**Operation**

Returns true if the current Rational DOORS user has create access for attribute definition in Module `m`. 
Attribute types

This section defines the functions that manipulate the types of Rational DOORS attributes. The following types are used: AttrType, which is a handle on an attribute type; and AttrBaseType, which is a handle on an attribute type’s base type.

AttrBaseType can have the following values:

<table>
<thead>
<tr>
<th>Scalar</th>
<th>Ranged</th>
<th>attrDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranged</td>
<td>attrInteger</td>
<td></td>
</tr>
<tr>
<td>Ranged</td>
<td>attrReal</td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td>attrText</td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td>attrString</td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td>attrUsername</td>
<td></td>
</tr>
</tbody>
</table>

Aggregate

| Enumeration | attrEnumeration |

They are used for determining the base type of an attribute type, for example, you might have an attribute type called "1 to 10" whose base type is an integer but has limits of 1 and 10.

Ranged types can have a maximum and minimum value.

Attribute type properties

Properties are defined for use with the . (dot) operator and an attribute type handle to extract information from an attribute type, as shown in the following syntax:

(AttrType at).property

The following tables list the properties and the information they extract:

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of an attribute type.</td>
</tr>
<tr>
<td>strings[n]</td>
<td>provides access to the names of an enumerated attribute type; the nth element (counting from 0). In the Rational DOORS user interface, this is the ‘value’ of the enumerated type.</td>
</tr>
<tr>
<td>description</td>
<td>The description of the attribute type</td>
</tr>
<tr>
<td>description[s]</td>
<td>The descriptions of the values in an enumerated type.</td>
</tr>
</tbody>
</table>
### Boolean property | Extracts
---|---
**canWrite** | Whether the user can delete the attribute type.
**system** | Whether the attribute type is system defined.

### Integer property | Extracts
---|---
**colors[n]** | The \( n \)th element (counting from 0) of the array of colors that are used in an enumeration attribute type.
**colours[n]** | The \( n \)th element (counting from 0) of the array of colors that are used in an enumeration attribute type.
**maxValue** | The maximum value for an attribute type or tests for the presence of a maximum value. Can also be of type `Real` or `Date`.
**minValue** | The minimum value for an attribute type or tests for the presence of a minimum value. Can also be of type `Real` or `Date`.
**size** | The number of elements of an enumerated type.
**values[n]** | The \( n \)th element (counting from 0) of the array of values used in an enumeration attribute type. In the Rational DOORS user interface, this is the ‘related number’ of the enumerated type.

### Any type property | Extracts
---|---
**type** | The base type of an attribute type.

### Example

```dxl
// name
AttrType at = find(current Module, "Created Thru")
print at.name  // prints  "Created Thru"

// type
AttrType at = find(current Module, "Integer")
print stringOf at.type
print at.type "\n"
// test for a minimum value
AttrType at = find(current Module, "Type with Min Int value")
```
if (at.minValue) {
   // Enter here if type has a minimum value.
   // The following is valid only if base type
   // is integer.
   // The operator is also defined for real and
   // date
   int i = at.minValue
}

// strings
AttrType t
t=find(current Module, "TableType")
print t.strings[1]

// size
AttrType at = find(current Module, "Boolean")
print at.size // prints "2"

// names
AttrType at = find(current Module, "Boolean")
print at.strings[0]
print at.strings[1]

// values
AttrType at = find(current,"Boolean")
print at.values[0]
print at.values[1]

// colors
AttrType at = find(current,"Boolean")
print at.colours[0]
print at.colors[1]

// canWrite and system
AttrType at
Module m = current
for at in m do{
   print at.name "= system: " at.system"; can
   write: " at.canWrite "\n"}
Concatenation (attribute base type)

The space character is the concatenation operator, which is shown as `<space>` in the following syntax:

```
AttrBaseType abt <space> string s
```

Concatenates the string `s` onto the attribute base type `abt` and returns the result as a string.

---

find(attribute type)

**Declaration**

```
AttrType find(Module m,
              string typeName)
```

**Operation**

Returns an attribute type handle for the attribute type named `typeName` in the module `m`, or null if the type does not exist.

**Example**

```
AttrType at = find(current Module, "Boolean")
if (null at)
    print "Failed\n"
```

---

isRanged

**Declaration**

```
bool isRanged(AttrType attrType)
```

**Operation**

Returns `true` if `attrType` is a range (can take minimum and maximum values). Otherwise, returns `false`.

---

isUsed

**Declaration**

```
bool isUsed(AttrType attrType)
```

**Operation**

Returns `true` if `attrType` is in use, in which case, its base type cannot be changed. Otherwise, returns `false`. For information on changing an attribute type's base type, see the `modify(attribute type)` function.
print(attribute base type)

Declaration
void print(AttrBaseType abt)

Operation
Prints the attribute base type \textit{abt} in the DXL Interaction window’s output pane.

stringOf(attribute base type)

Declaration
string stringOf(AttrBaseType abt)

Operation
Returns attribute base type \textit{abt} as a string.

getRealColorOptionForTypes

Declaration
bool getRealColorOptionForTypes()

Operation
Returns \texttt{true} if the values contained within the color array of an \texttt{AttrType} are real color identifiers. Returns \texttt{false} if the values are logical color identifiers (the default).

setRealColorOptionForTypes

Declaration
void setRealColorOptionForTypes(bool realColors)

Operation
If \texttt{realColors} is \texttt{true}, sets the values contained within the color array of an \texttt{AttrType} to real color identifiers. If \texttt{realColors} is \texttt{false}, sets the values to logical color identifiers (the default).

\textbf{Note:} The functions that create and modify an \texttt{AttrType} expect arrays of real colors as arguments. Therefore, prior to any calls being made to either create\,(attribute\ type) or modify\,(attribute\ type), this function must be called setting \texttt{realColors} to \texttt{true}. 
setDescription

Declaration
AttrType setDescription(AttrType at, string desc, string &errMess)

Operation
Sets the description for the specified attribute type. Returns null if the description is not successfully updated.

setURI

Declaration
AttrType setURI(AttrType at, string URI, string &errMess)
AttrType setURI(AttrType at, string name, string URI, string &errMess)
AttrType setURI(AttrType at, int index, string URI, string &errMess)

Operation
Sets the URI for the specified attribute type. Returns a modified attribute type. If there is an error, the message is returned in the final string parameter. The URI can be set for a specified enumeration value or enumeration index.

Example
AttrType at
string errorMsg
string index[] = { "first", "second", "third" }  
at = setURI(at, "http://www.webaddress.com", errorMsg)
at = setURI(at, index[0], "http://www.webaddress.com", errorMsg)

getURI

Declaration
string uri(AttrType at)
string uri(AttrType at, string name)
string uri(AttrType at, int index)

Operation
Gets the URI for the specified attribute type or for a named enumeration value or for a enumeration index.
for attribute type in module

Syntax
for at in Module m do {
    ...
}
where:

    at          is a variable of type AttrType
    m           is a variable of type Module

Operation
Assigns the variable at to be each successive attribute type definition found in module m.

Example
AttrType at
for at in current Module do {
    print at.name "\n"
}

Attribute type access controls

This section describes functions that report on access rights for an attribute type.

canCreate(attribute type)

Declaration
bool canCreate(AttrType attrType)

Operation
Returns true if the current Rational DOORS user has create access to the attribute type attrType.

canControl(attribute type)

Declaration
bool canControl(AttrType attrType)
Operation
Returns true if the current Rational DOORS user can change the access controls on the attribute type attrType.

canModify(attribute type)

Declaration
bool canModify(AttrType attrType)

Operation
Returns true if the current Rational DOORS user can modify the attribute type attrType.

canRead(attribute type)

Declaration
bool canRead(AttrType attrType)

Operation
Returns true if the current Rational DOORS user can read the attribute type attrType.

canDelete(attribute type)

Declaration
bool canDelete(AttrType attrType)

Operation
Returns true if the current Rational DOORS user can delete the attribute type attrType. Otherwise, returns false.

canCreateAttrTypes

Declaration
bool canCreateAttrTypes(Module m)

Operation
Returns true if the current Rational DOORS user has create access for attribute types in Module m.

Attribute type manipulation

This section defines functions for creating new attribute types, modifying, and deleting them.
To modify an attribute type, the user must have modify access to it (the canWrite property returns true). No changes can be made in edit shareable mode or read-only mode. System types cannot be edited (the system property returns true). For information on properties, see “Attribute type properties,” on page 452.

create(attribute type)

Declaration

AttrType create(string name,
     AttrBaseType abt,
     string &errmess)

AttrType create(string name,
        {int|real|Date} min,
        {int|real|Date} max,
        string &errmess)

AttrType create(string name,
        string codes[ ],
          [int values[ ],]
          [int colors[ ],]
          [string descs[ ],]
          [string URI[ ],]
        string &errmess)

Operation

If the operation fails, all forms of create return an error message in errmess.
The function also throws a run-time DXL error for an invalid input, for example a duplicate type name. These errors can be trapped using lastError and noError.
The first form creates a new attribute type, of name name and base type abt.
The next form creates a new attribute type named name, of base type int, real or Date, for a range of min to max.
The last form creates enumeration types named name, using enumeration names codes, with optional values values, colors colors, descriptions descs, and URI URI. The argument URI[ ] is the URI for each value.

Note: This function expects arrays of real colors as arguments. Therefore, prior to any calls being made to create, the setRealColorOptionForTypes function must be called setting realColors to true.

Example

    // basic create
    string errmess = ""
    AttrType at = create("Cost", attrInteger, errmess)
    if (!null errmess)
        print "Attribute type creation failed\n"
// create enumeration type
string names[] = {"Tested", "Under Test", "Not Tested"}
int values[] = {1,2,3}
int colors[] = {-1, 20, 14}
string mess = ""
AttrType at = create("Test Status", names, values, colors, mess)
if (!null mess)
    print "Type creation failed\n"

delete(attribute type)

Declaration
bool delete(AttrType at,
            string &errmsg)

Operation
Deletes the AttrType whose handle is at. If the operation fails, returns an error message in errmsg.

modify(attribute type)

Declaration
AttrType modify(AttrType type,
                string newName
                [, string codes[ ],
                int values[ ],
                int colors[ ],
                string desc[ ],
                string URI[ ],
                [, int arrMaps[ ]],
                string &errmsg)

AttrType modify(AttrType type,
                AttrBaseType new,
                string &errmsg)

Operation
The first form, without any optional parameters, changes the name of the specified attribute type to newName. If supplied, codes, values, colors, desc, and URI modify those properties of an existing enumerated type. The argument URI[] is the URI for each enumerated type. In the user interface, the term values maps to codes, and the term related numbers maps to values. If the type is being used by an attribute, colors cannot be added where they were not previously assigned, and, arrMaps must be supplied in order to map old values to the new ones.
The second form changes the base type of the specified attribute type. If \textit{type} is in use the call fails.

\textbf{Note:} Color numbers now refer to real colors rather than logical colors. Enumerated attribute types in Rational DOORS 4 have their colors translated during migration.

For all forms, the \texttt{errmess} argument is currently not used, but is reserved for future enhancements. You can trap errors using \texttt{lastError} and \texttt{noError}.

\textbf{Note:} This function expects arrays of real colors as arguments. Therefore, prior to any calls being made to \texttt{modify}, the \texttt{setRealColorOptionForTypes} function must be called setting \texttt{realColors} to \texttt{true}.

\textbf{Example}

```dxl
//This example adds "Invalid Test" to the end of the list of possible enumeration values, leaving the remaining value intact.

AttrType modifyAndAdd(AttrType atTypeToEdit, string sTypeName, string arrValues[], int arrOrdinals[], int arrColours[], string& sErrMsg)
{
    int arrMaps[atTypeToEdit.size + 1]
    int i

    for (i = 0; i < atTypeToEdit.size + 1; i++) {
        arrMaps[i] = i;
    }

    return modify(atTypeToEdit, sTypeName, arrValues, arrOrdinals, arrColours, arrMaps, sErrMsg)
}
```

```dxl
setRealColorOptionForTypes(true)
AttrType at = find(current Module, "Test Status")

string new_strings[at.size+1]
int new_values[at.size+1]
int new_colors[at.size+1]

int i=0
for(i = 0; i < at.size; i++)
{
    new_strings[i] = at.strings[i]
```
string errmsg
new_strings[at.size] = "highest"  // This is name of new value for type
new_values[at.size] = at.size
new_colors[at.size] = -1

string at_name = at.name
AttrType at_new = modifyAndAdd(at, at_name, new_strings, new_values, new_colors, errmsg)

AttrType modifyAndAdd(AttrType atTypeToEdit, string sTypeName, string arrValues[], int arrOrdinals[], int arrColours[], string& sErrMsg)
{
    int arrMaps[atTypeToEdit.size + 1]
    int i

    for (i = 0; i < atTypeToEdit.size + 1; i++) {
        arrMaps[i] = i;
    }
    return modify(atTypeToEdit, sTypeName, arrValues, arrOrdinals, arrColours, arrMaps, sErrMsg)
}

setRealColorOptionForTypes(true)
AttrType at = find(current Module, "Test Status")

string new_strings[at.size+1]
int new_values[at.size+1]
int new_colors[at.size+1]

int i=0
for(i = 0; i < at.size; i++)
new_strings[i] = at.strings[i]
new_values[i] = at.values[i]
new_colors[i] = at.colors[i]

string errmsg
new_strings[at.size] = "Invalid Test"               // This is name of new value for type
new_values[at.size] = at.size
new_colors[at.size] = -1

string at_name = at.name
AttrType at_new = modifyAndAdd(at, at_name, new_strings, new_values, new_colors, errmsg)

setMaxValue

Declaration

bool setMaxValue(AttrType type,
                {int|real|Date} maxValue,
                bool maxApplies)

Operation

Specifies a maximum value for the ranged attribute type type, provided maxApplies is true.
If the call succeeds, returns true; otherwise, returns false. If maxApplies is false, the maximum value is ignored,
and the function returns true.
If the specified maximum value is less than the minimum value, the call fails.
If the specified type is not a ranged type, or is not of the same type as maxValue, a run-time error occurs, which can be
trapped using lastError and noError.

Example

Module m = current
AttrType atype = find(m,"MyType")
string sBaseType = stringOf(atype.type)
if(sBaseType == "Integer")
{
    // set a maximum of 100, and enable the
    // maximum
setMaxValue(atype,100,true)
}

setMinValue

Declaration
bool setMinValue(AttrType type,
    {int|real|Date} minValue,
    bool minApplies)

Operation
Specifies a minimum value for the ranged attribute type type, provided minApplies is true.
If the call succeeds, returns true; otherwise, returns false. If minApplies is false, the maximum value is ignored, and the function returns true.
If the specified maximum value is less than the minimum value, the call fails.
If the specified type is not a ranged type, or is not of the same type as minValue, a run-time error occurs, which can be trapped using lastError and noError.

Example
Module m = current
AttrType atype = find(m,"MyType")
string sBaseType = stringOf(atype.type)
if(sBaseType == "Integer")
{
    //set a minimum of 10, and enable the minimum
    setMinValue(atype,10,true)
}

DXL attribute

DXL attribute is an option on the Define Attribute window, which enables you to write a DXL program that calculates the value of the attribute being defined. The calculation only takes place the first time the attribute is accessed, or if it is later cleared to null and is subsequently accessed again. This means the DXL code is not executed when the containing module is opened, but when some event occurs that causes the attribute to be accessed. For example, the event could be because the attribute is being displayed in a column, or because the user opens the Formal Object Editor window on an object with a DXL attribute value.

DXL attribute provides a means of initializing an attribute using DXL, and then caching that value so that subsequent attribute access does not involve recalculation. If the code resets the attribute to the null string, recalculation occurs on the next access. Just setting the value to the null string is not sufficient to invoke recalculation. The attribute value must be accessed after the reset to null, for a recalculation to take place.
For example, if attribute "Outgoing" is displayed in a Rational DOORS column, the initial value is calculated for each "Outgoing" attribute as the user views it. If more outgoing links are created, the attribute values do not change; to do this, the recalculation must be forced, possibly from another DXL application that contains the following script fragment:

```dxl
Object o
for o in current Module do
  o."Outgoing" = (string null)
```

The (string null) ensures a null value, as compared to the integer 0, or the empty string "".

**Note:** The perm void refresh(Module m) should not be used in DXL attributes.

### `attrDXLName`

#### Declaration

Object `obj`

const string `attrDXLName`

#### Operation

DXL attribute programs run in a context where the variable `obj` is already declared to refer to the object whose attribute is being calculated.

The constant `attrDXLName` can be used instead of a literal attribute name to refer to the attribute value that is being calculated. This enables one piece of DXL attribute to be used for several attributes without being modified.

#### Example

```dxl
obj.attrDXLName = today
```

### DXL attribute example program

This example in `$DOORSHOME/lib/dxl/attrib/impact.dxl`:

```
// impact.dxl -- example of DXL attribute
/*
 DXL attribute provides a means of initializing an attribute using DXL, so that subsequent accesses of the attribute do not involve re-calculation.
 This example of DXL attribute requires that an integer attribute named "Outgoing" exists and has been defined with the Rational DOORS GUI to use this file as its DXL value.
 "Outgoing" is set to the number of links leaving its object.
 */

Link l
```
int count = 0
// obj is the predeclared object whose attribute
// we are calculating
for l in obj->"*" do count++
// count outgoing links
obj."Outgoing" = count
// initialize the cached value
// resetting to (string null)
// in a DXL program
// will force re-calculation
// end of impact.dx1
Chapter 21
Access controls

This chapter describes access controls:

- Controlling access
- Locking
- Example programs

Controlling access

This section defines properties, operators, functions and for loops that work with access controls. Many of these elements use the data types Permission and AccessRec.

Properties

The following properties of type Permission are used for setting access controls, using the assignment operator.

- none
- read This is automatically given for modify, create, delete, or control.
- create Automatically confers read access. Automatically given for control.
- modify Automatically confers read access. Automatically given for control.
- delete Automatically confers read and modify access.
- control Automatically confers read, modify and create access.
- write This is a bitwise OR of modify, create and delete; it is only supported for compatibility with earlier releases.
- change Identical to control, this is only supported for compatibility with earlier releases.

Operators

As with other data types, the assignment operator = is used to set a permission, as shown in the following syntax:

Permission $p$ = permission
where:

\[ p \] is a variable of type Permission

\[ \text{permission} \] is a variable of type Permission

The | (pipe) operator performs bitwise OR operations on permissions as shown in the following syntax:

\[ \text{Permission } x \mid \text{Permission } y \]

The & operator performs bitwise AND operations on permissions as shown in the following syntax:

\[ \text{Permission } x \& \text{Permission } y \]

The == relational operator performs comparison on permissions as shown in the following syntax:

\[ \text{Permission } x == \text{Permission } y \]

**Example**

Permission all = read|create|modify|delete|control

---

**Access status**

**Declaration**

bool read(AccessRec ar)  
bool create(AccessRec ar)  
bool modify(AccessRec ar)  
bool delete(AccessRec ar)  
bool control(AccessRec ar)  
bool write(AccessRec ar)  
bool change(AccessRec ar)

**Operation**

Each of the first five functions returns true if the access record confers modify, create, delete, control, or read permission. Both write and change are supported for compatibility with earlier releases; write returns true if the access record confers modify permission, and change returns true if the access record confers control permission. If the specified permission is not present, each function returns false.

**Note:** When using these functions with groups, any information returned for create permission is redundant as there is no create permission on groups.

---

**partition**

**Declaration**

bool partition(AccessRec ar)
**Operation**

Returns `true` if the data that is associated with the access record has been partitioned out.

---

**get, getDef, getVal**

**Declaration**

```dxl
AccessRec get({Object o|Module m|Project p|
    Folder f|Item i|View v|Group g},
    [AttrType at,]
    {string user|string group,
    string &message)
```

```dxl
AccessRec get{Def|Val}(Module m,
    AttrDef ad,
    {string user|string group},
    string &message)
```

**Operation**

The first form returns the access record for object `o`, module `m`, project `p`, folder `f`, item `i`, view `v`, or group `g` for Rational DOORS user with name `user`, or group with name `group`. Optionally, for a module, the access record can be for a specific attribute type `at`.

The function `getDef` returns the access record for the attribute definition `ad` in module `m`.

The function `getVal` returns the access record for the attribute value of the attribute definition `ad` in module `m`.

For all these functions, the strings `user` or `group`, are the Rational DOORS user or group, to whom the access record applies. If they are null, the function returns the default access record. If the operation succeeds, returns a null string in `message`; otherwise, returns an error message.

If no specific access control setting has been made, these functions return null. However, a parent object or module setting might be being inherited.

---

**getImplied**

**Declaration**

```dxl
string getImplied({Object o|Module m|Project p|Folder f|Item i}, Permission &ps)
```

**Operation**

Returns the permissions that are inherited by children of the resource when the user has create permission to the resource (extra access propagated by create).

Returns the permissions inherited by children of object `o`, module `m`, folder `f`, item `i`, or view `v`. Optionally, when specifying a module, the permissions can be for a specific attribute type `at`.

If the operation succeeds, returns a null string; otherwise, returns an error message.
If no specific extra access setting has been made, these functions return null. However, a parent object or module setting might be being inherited.

### inherited, inheritedDef, inheritedVal

#### Declaration

```dxl
string inherited({Object o|Module m|Project p|Folder f|Item i|View v} [,AttrType at])
```

```dxl
string inherited{Def|Val}(Module m, AttrDef ad)
```

#### Operation

These functions set access control to be inherited rather than specific.

The first form does this for object `o`, module `m`, project `p`, folder `f`, item `i`, or view `v`. Optionally, for a module, the access record can be for a specific attribute type `at`.

The `inheritedDef` function does this for the attribute definition `ad` in module `m`. The `inheritedVal` function does it for the attribute value of the attribute definition `ad` in module `m`.

If the operation succeeds, returns `null`; otherwise, returns an error message.

### isAccessInherited

#### Declaration

```dxl
string isAccessInherited({Object o|Project p|Folder f|Item i|View v}, bool &inherited)
```

```dxl
string isAccessInherited(Module m,[AttrType at,]bool &inherited)
```

```dxl
string isAccessInherited{Def|Val}(Module m, AttrDef ad, bool &inherited)
```

#### Operation

Returns whether the access rights are inherited.

The first form does this for object `o`, project `p`, folder `f`, item `i`, or view `v`.

The second form does this for module `m`. Optionally, the access record can be for a specific attribute type `at`.

The `isAccessInheritedDef` function does this for the attribute definition `ad` in module `m`. The `isAccessInheritedVal` function does it for the attribute value of the attribute definition `ad` in module `m`.

If the operation succeeds, returns `null`; otherwise, returns an error message.
isDefault

**Declaration**

```dxl
bool isDefault(AccessRec ar)
```

**Operation**

Returns `true` if `ar` is the default access record for a particular item; otherwise, returns `false`.

**Example**

```dxl
AccessRec ar

// process module (exclude inherited rights)
for ar in current Module do
{
    // only relevant if default
    if (isDefault(ar) == true)
    {
        // .. do stuff
    }
}
```

set, setDef, setVal

**Declaration**

```dxl
string set({Object o|Module m|Project p|Folder f|Item i|View v|Group g},
            [AttrType at,]
            Permission ps,
            {string user|string group})
```

```dxl
string set{Def|Val}(Module m,
                AttrDef ad,
                Permission ps,
                {string user|string group})
```

**Operation**

The first form sets permission `ps` on object `o`, module `m`, project `p`, folder `f`, item `i`, view `v` or Group `g`, for Rational DOORS user with name `user`, or group with name `group`. Optionally, for a module, the permission can be for a specific attribute type `at`.

The function `setDef` sets the permissions for the access list of the attribute definition `ad` in module `m`.

The function `setVal` sets the permission of all values of the attribute definition `ad` in module `m`.

For all these functions, if `user/group` is null, the function modifies the default access control. If the operation succeeds, it returns a null string; otherwise, it returns an error message. When retrieving access for an item and the user/group name retrieved is being assigned to a string, ensure that an empty string is appended to the end of the assigned string.
In some circumstances it might be possible to add the administrator user to a Rational DOORS access list. This should be guarded against.

**Example**

```
set(current Object, read|modify|delete|control, doorsname)
```

---

### setImplied

**Declaration**

```
string setImplied({Object o|Module m|Project p|Folder f|Item i},
                  Permission ps)
```

**Operation**

Sets the extra access control propagated by create for children of the resource.

Sets permission `ps` on object `o`, module `m`, project `p`, folder `f`, item `i`, or view `v`.

If the operation succeeds, returns a null string; otherwise, returns an error message.

---

### specific, specificDef, specificVal

**Declaration**

```
string specific({Object o|Module m|Project p|Folder f|Item i|View v},
                [AttrType at])
```

```
string specific{Def|Val}(Module m, AttrDef ad)
```

**Operation**

These functions set access control to be specific rather than inherited. The item is left with specific access rights, which are identical to the inherited rights at the time the function is called. These functions have no effect if the access rights are already specific.

The first form does this for object `o`, module `m`, project `p`, folder `f`, item `i`, or view `v`. Optionally, for a module, the access rights can be for a specific attribute type `at`.

The `specificDef` function does this for the attribute definition `ad` in module `m`. The `specificVal` function does it for the attribute value of the attribute definition `ad` in module `m`.

If the operation succeeds, returns `null`; otherwise, returns an error message. If the user does not have control access, the call fails.
unset, unsetDef, unsetVal, unsetAll

Declaration

string unset({Object o|Project p|Module m| Folder f|Item i|View v|Group g},
       [AttrType at,]
       {string user|string group})

string unset{Def|Val}(Module m,
       AttrDef ad,
       {string user|string group})

string unsetAll({Object o|Project p|Module m| Folder f|Item i|View v|Group g},
       [AttrType at,])

string unsetAll{Def|Val}(Module m, AttrDef ad)

Operation

The first form clears the permission set on object o, project p, folder f, item i, View v, or Group g for Rational DOORS user with name user, or group with name group.

The second form clears the permission set on module m. Optionally, clears the permission for a specific attribute type at.

The function unsetDef clears the permissions set for the access list of the attribute definition ad in module m.

The function unsetVal clears the permissions set for all values of the attribute definition ad in module m.

The function unsetAll clears all user permissions set for the specified argument.

The function unsetAllDef clears user permissions set for the access list of the attribute definition ad in module m.

The function unsetAllVal clears user permissions set for all values of the attribute definition ad.

If user (or group) is null, the call fails. If the operation succeeds, returns the null string; otherwise, returns an error message.

Note: Care should be taken when using these perms. The unsetting of the access controls is immediate, so if the user is removing specific access controls for an item, they must ensure that the default user has control access before use. Furthermore, care should be taken when using these perms in loops.

Example

Module m = current
string err = unset(m, "joe")
if (!null err){
   infoBox(err)
}

475
username

Declaration
string username(AccessRec a)

Operation
Returns the user name associated with the access record a. A null result means that access record a is the default record.

Example
string mess
AccessRec a = get(current Object, null, mess)
if (null mess) {
   if (null a) {
      print "default record"
   } else {
      print (username a) "\n"
   }
} else {
   print "error getting access record: " mess
}

for access record in type

Syntax
for ar in type do {
   ...
}
where:

ar is a variable of type AccessRec
type is a variable of type Module, Object, Folder, Item, View, AttrDef, Group or AttrType

Operation
Assigns the variable ar to be each successive access record in type, excluding inherited access rights.

Example
AccessRec ar
for ar in current Object do {
   string user = username ar
if (null user) {
    print "default"
} else {
    print user
}

print " can read? " (read ar) "\n"

for access record in all type

Syntax
for ar in all type do {
    ...
}

where:

ar is a variable of type AccessRec
type is a variable of type Module, Object, Folder, Item, View, AttrDef, or AttrType

Operation
Assigns the variable ar to be each successive access record in type, including inherited access rights.

for access record in values

Syntax
for ar in values(AttrDef ad) do {
    ...
}

where:

ar is a variable of type AccessRec
ad is a variable of type AttrDef

Operation
Assigns the variable ar to be each successive record found for the list of attribute values obtained by passing the attribute definition ad to the function values.

Example
AccessRec ar
AttrDef ad = find(current, "Object Heading")
for ar in values ad do {
    print (username ar) " can read " (read ar) "\n"
}

**Locking**

This topic defines functions that are used in conjunction with access controls to implement shared access to modules.

In the context of access control, a section is defined as anything with a specific access control, along with everything that inherits that access control.

The lock manager functions are described in “Locking,” on page 931.

---

### isLockedByUser

**Declaration**

```dxl
bool isLockedByUser(Object o)
```

**Operation**

Returns `true` if the specified object is locked by the current user when in edit shareable mode. Otherwise, returns `false`.

This function is not equivalent to checking whether the current user can modify the given object.

---

### lock(object)

**Declaration**

```dxl
string lock(Object o)
```

**Operation**

Locks object `o`. If the operation succeeds, returns `null`; otherwise, returns an error message.

This function only makes sense when `o` is in a module that has been opened shareable.

**Example**

```dxl
if (isShare current) {
    string mess = lock current Object
    if (!null mess)
        print "lock failed: " mess "\n"
}
```
Unlock object functions

Declaration

bool unlockDiscard{All|Section}(Object o)
bool unlockSave{All|Section}(Object o)

Operation

These functions unlock sections. The functions `unlockDiscardAll` and `unlockSaveAll` unlock all sections in the module containing `o`. The functions `unlockDiscardSection` and `unlockSaveSection` unlock the section containing `o`.

The functions either discard changes or save changes before unlocking according to the function name.

If the operation is successful, returns `true`; otherwise, returns `false`.

Example programs

This section contains two example programs.

Setting access control example

This example shows how to set the default specific access rights, assuming the calling user has permission so to do.

// access control setting example
/

Example Access control setting program.
Sets all objects in the current display set
(i.e. respecting filtering, outlining, level,
etc.) to have a specific access control, thus
enabling them to be locked in shareable mode.
Current module must be editable, and is then
reopened shareable.
*/

if (null current Module) {
    ack "Please run this program from a module"
    halt
} else if (!isEdit current) {
    ack "current module must be editable to set permissions"
    halt
} else if ((level current Module)==0) {
    ack "Please set a specific level display\n"  "all objects at this level will be made\n"
"lockable by giving them a specific
default\n" //-
"access control"

halt
}

Object o
string modName = (current Module)."Name"

for o in current Module do {
string err
    if (level o != level current Module)
        // just make selected level lockable
        continue
    // alter the default ACL record
    err = set(o,read|modify|delete|control,null)
    if (!null err) {
        ack "problem setting default ACL: " err
        halt
    }
}

save current // save our work

if (close current)
    share modName
    // open with new lockable sections

---

**Reporting access control example**

The following program illustrates some more access control features:

// access control example
/
*  
Example Access Control DXL
*/

if (null current Module) {
    ack "Please run this program from a module"
    halt
}

// function to display an ACL record:
bool showAcl(string user, AccessRec acl, string type) {
    string thisuser = (username acl)
if (thisuser != user) return false
print "User: " user " has "
bool something = false
if (read acl) {
    something = true
    print "read "
}
if (modify acl) {
    something = true
    print "modify "
}
if (delete acl) {
    something = true
    print "delete "
}
if (control acl) {
    something = true
    print "control "
}
if (!something) print "no "
print "powers on " type "\n"
return true
}
string user = doorsname
AccessRec acl
bool found = false
for acl in current Module do {
    if (showAcl(user, acl, "current module")) {
        found = true
        break
    }
}
if (!found)
    print "default permission in current module\n"
found = false
for acl in current Object do {
    if (showAcl(user, acl, "current object")) {
        found = true
        break
    }
}
if (!found)
    print "default power on current object\n"

string fail

fail = set(current Module, change, user)

if (!null fail)
    print "Setting change failed for current
    module: " fail "\n"
Chapter 22
Dialog boxes

This chapter describes DXL facilities for creating Rational DOORS dialog boxes, which are any windows that are constructed by DXL. Throughout this manual, the term dialog box is used to mean Rational DOORS dialog box. This chapter covers the following facilities:

- Icons
- Message boxes
- Dialog box functions
- Dialog box elements
- Common element operations
- Simple elements for dialog boxes
- Choice dialog box elements
- View elements
- Text editor elements
- Buttons
- Canvas
- Complex canvases
- Toolbars
- Colors
- Simple placement
- Constrained placement
- Progress bar
- DBE resizing
- HTML Control
- HTML Edit Control

An extensive example of all dialog box functions can be found in ddbintro.dx1 in the DXL example directory.

Icons

This section defines constants and functions for using icons within dialog boxes. The functions use the Icon data type.
Constants

Declaration

Icon iconDatabase
Icon iconProject
Icon iconProjectCut
Icon iconProjectDeleted
Icon iconProjectOpen
Icon iconProjectOpenDeleted
Icon iconFormal
Icon iconFormalCut
Icon iconFormalDeleted
Icon iconLink
Icon iconLinkCut
Icon iconLinkDeleted
Icon iconDescriptive
Icon iconDescriptiveCut
Icon iconDescriptiveDeleted
Icon iconFolder
Icon iconFolderCut
Icon iconFolderDeleted
Icon iconFolderOpen
Icon iconFolderOpenDeleted
Icon iconDatabase
Icon iconGroup
Icon iconGroupDisabled
Icon iconUser
Icon iconUserDisabled
Icon iconReadOnly
Icon iconNone
Icon iconAuthenticatingUser
Operation

These standard icon values can be used in functions where a value of type Icon is required. Icon constants starting folder are for tool bars; those starting icon are for list and tree views. Use the set(icon) function to specify an icon. Use the same function with iconNone to remove an icon. You can also load icons from disk. For further information, see the load function.

Example

set(theTab, 0, iconDatabase)

---

load

Declaration

Icon load(string fileName)

Operation

Loads an icon from disk. The fileName argument must be a full path.

For Windows platforms, if the file has an extension .ico, Rational DOORS assumes it is a Windows icon file; otherwise, Rational DOORS assumes it is a Windows bitmap.

Masks only work with icon files, not with bitmaps. An icon file should represent an image of either 16x16 or 32x32 pixels. The file should have no more than 8 bits per pixel (256 colors).

On UNIX platforms, icon files are .xpm (X PixMap) files; icons are Motif Pixmaps. For further information, see the XPM documentation.

Example

Icon i = load("c:\\test.ico")
set(theTab, 0, i)

destroy(icon)

Declaration

void destroy(Icon iconName)

Operation

Frees up resources used by iconName. Use this when you destroy a dialog box; for further information, see the destroy(dialog box) function.

Example

Icon ic = load("c:\\test.ico")
set(theTab, 0, ic)
// .. then on program close
destroy ic
Message boxes

This section defines functions that create message boxes. Message boxes provide a convenient way of informing users of events, such as confirmations or errors. The functions use the DB data type.

**acknowledge**

**Declaration**

```dxl
void acknowledge([DB box,] 
    string message)
```

**Operation**

Pops up a message box containing the message and an Acknowledge or OK button, depending on platform, in a manner compatible with the rest of Rational DOORS. Execution of the DXL program is suspended until the user clicks Acknowledge or OK.

The optional DB box argument positions the message box over a specific dialog box.

**Example**

```dxl
ack "Invalid weight supplied for grommet"
```

**errorBox**

**Declaration**

```dxl
void errorBox([DB box,] 
    string message)
```

**Operation**

Pops up a message box containing the error and an Acknowledge or OK button, depending on platform, in a manner compatible with the rest of Rational DOORS. Execution of the DXL program is suspended until the user clicks Acknowledge or OK.

The optional DB box argument positions the message box over a specific dialog box.

**Example**

```dxl
errorBox "Path does not exist"
```

**infoBox**

**Declaration**

```dxl
void infoBox([DB box,] 
    string message)
```
Operation

Pops up a message box containing information and an **Acknowledge** or **OK** button, depending on platform, in a manner compatible with the rest of Rational DOORS. Execution of the DXL program is suspended until the user clicks **Acknowledge** or **OK**.

The optional `DB box` argument positions the message box over a specific dialog box.

Example

```
infoBox "Insufficient space on specified drive"
```

---

**warningBox**

Declaration

```dxl
void warningBox([DB box,]
               string message)
```

Operation

Pops up a message box containing the warning and an **Acknowledge** or **OK** button, depending on platform, in a manner compatible with the rest of Rational DOORS. Execution of the DXL program is suspended until the user clicks **Acknowledge** or **OK**.

The optional `DB box` argument positions the message box over a specific dialog box.

Example

```
warningBox "This deletes all files - continue?"
```

---

**confirm**

Declaration

```dxl
bool confirm([DB box,]
             string message,
             int severity)
```

Operation

Pops up a confirmation box containing the message and buttons labeled **Confirm** and **Cancel**. The `severity` argument controls the icon displayed in the message box; the value can be one of `msgInfo` (blue i), `msgWarning` (red X), `msgError` (yellow !), or `msgQuery` (black ?). The DXL program is suspended until the user clicks one of the buttons, when the function returns `true` for **Confirm** and `false` for **Cancel**.

**Note:** The use of `\t` within the `message` string is not supported.

The optional `box` argument positions the message box over a specific dialog box.

Example

```
if (confirm "Delete all records?")
    deleteRecords
```
**query**

**Declaration**

```dxl
tuple query([DB box,]
    string message,
    string[] buttons)
```

**Operation**

Displays a message box with the message and buttons with the labels provided in the string array. The DXL program is halted until the user clicks one of the buttons, when the function returns with the index for that button.

The optional `DB box` argument positions the message box over a specific dialog box.

**Example**

```dxl
string analyopts[] = {"Linear", "Quadratic", "Spline"}
int mode = query("Select analysis model", analyopts)
if (mode == 0) {
    doLinearAnalysis
} else if (mode == 1) {
    doQuadraticAnalysis
} else {
    doSplineAnalysis
}
```

**messageBox**

**Declaration**

```dxl
tuple messageBox([DB box,]
    string message,
    string buttons[],
    int severity)
```

**Operation**

Displays a message box with the message, and buttons with the labels provided in the string array. The `severity` argument controls the title of the message box; the value can be one of `msgInfo` (blue i), `msgWarning` (red x), `msgError` (yellow !), or `msgQuery` (black ?). The DXL program is halted until the user clicks one of the buttons, when the function returns with the index for that button.

The optional `DB box` argument positions the message box over a specific dialog box.

**Example**

```dxl
string buttons[] = {"Yes", "No", "Cancel"}
```
int answer = messageBox("Do you want save?", buttons, msgQuery)

print answer
confirm("Really?", msgWarning)

---

Dialog box functions

This section defines functions for dialog boxes, which are built around the data type DB. Dialog boxes contain elements, such as buttons, fields or labels, which are represented by the data type DBE.

addAcceleratorKey

Declaration

void addAcceleratorKey(DB db, void dxlCallback(), char accelerator, int modifierKeyFlags)

Operation

Adds an accelerator key accelerator to the dialog db with the callback function dxlCallback() and the passed-in modifierKeyFlags. modifierKeyFlags is used in conjunction with the accelerator parameter to change which key should be pressed with the accelerator key. Possible values for it are modKeyNone, modKeyCtrl, modKeyShift and null.

The specified DXL callback fn dxlCallback() executes for the specified keystroke combination being pressed when the DXL dialog box db is active.

Only call this perm after the dialog box db has been realized, otherwise a DXL run-time error will occur.

Example

void fn()
{
    print "callback fires\n"
}

DB db = create("testDialog", styleStandard)
realize db

// The callback fn() will be executed on pressing Shift+F7 when the dialog db is active.
addAcceleratorKey(db, fn, keyF7, modKeyShift)
baseWin

Declaration
void baseWin(DB box)

Operation
This function is only for use in batch mode.
Displays the dialog box and suspends execution of the DXL program. Execution continues in callbacks from the buttons on the dialog box. No code should be placed after a call to baseWin, because it would never be executed.

block

Declaration
void block(DB modalBox)

Operation
Displays a modal dialog box. When a modal dialog box is displayed, the rest of the Rational DOORS interface is insensitive, leaving only the given dialog box able to receive input. The interface remains in this state until either the dialog box is closed or the release function is called.
Unlike show, DXL program execution is resumed after the call to block when the modal dialog box is released.

Example
block importantQuesBox
processResult

busy

Declaration
void busy(DB box)

Operation
Sets the window busy, displaying the waiting cursor and making it insensitive to input. Use the ready function to reset the dialog box to normal.

Example
busy stressResultsBox
centered

Declaration

DB centered(string title)

Operation

Creates a dialog box that is centered on the screen. Nothing appears on the screen until it is passed to either the `block` or `show(dialog box)` function, when the dialog box window title bar contains `title`.

Example

DB splashBox = centered "Welcome to Example"

create(dialog box)

Declaration

DB create(
    [{Module|DB} parent,,] string title 
    [,int options])

Operation

Creates a new, empty dialog box structure. Nothing appears on the screen until it is passed as an argument to `show`, when the dialog box window title bar contains `title`.

The optional first argument creates a child window of the module or dialog box specified by `parent`. When a child window is hidden, its parent is put in front of any other windows. The optional third argument defines the style of the dialog box; it can have bitwise OR combinations of the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>styleStandard</td>
<td>Appears like other Rational DOORS windows.</td>
</tr>
<tr>
<td>styleFixed</td>
<td>Has no resizing capability.</td>
</tr>
<tr>
<td>styleCentered</td>
<td>Appears in the center of the screen.</td>
</tr>
<tr>
<td>styleCentred</td>
<td>Appears in the center of the screen.</td>
</tr>
<tr>
<td>styleFloating</td>
<td>Appears above all other Rational DOORS windows.</td>
</tr>
<tr>
<td>styleNoBorder</td>
<td>Has no title bar or resizing capability.</td>
</tr>
<tr>
<td>styleThemed</td>
<td>Inherits themed styles into tabs</td>
</tr>
<tr>
<td>styleAutoParent</td>
<td>Automatically set the parenting of controls based on layout information</td>
</tr>
</tbody>
</table>
Example

DB parseBox = create("Sim File Parser", styleCentered|styleFixed)
label(parseBox, "Nothing in here yet")
show parseBox

createButtonBar

See “createButtonBar,” on page 637.

createItem

See “createItem,” on page 637.

createCombo

See “createCombo,” on page 642.

destroy(dialog box)

Declaration

void destroy(DB box)

Operation

Frees up resources used by box. The specified box should not be used after it has been destroyed without being re-initialized. After using destroy, you should set box to null.

If the dialog box used icons, you should also destroy them using the load function.

Note: Destroy should not be used within a callback function for a DBE.

getPos

Declaration

void getPos(DB myWindow, int& x, int& y)

Operation

Returns in x and y the screen co-ordinates of the origin of the specified window.
getSize

Declaration
void getSize(DB myWindow,
        int& w,
        int& h)

Operation
Returns in h and w the height and width of the specified window. Dimensions are returned in pixels.

title

Declaration
string getTitle(DB myWindow)

Operation
Returns the title of the specified window.

getBorderSize

Declaration
int getBorderSize(DB myWindow)

Operation
Returns the width in pixels of the border for the specified dialog box.

Example
DB DBox = create("Dialog Box", styleCentered|styleFixed)
int i = getBorderSize(DBox)
label(DBox, "Border size is " i '')

    show DBox

captionHeight

Declaration
int getCaptionHeight(DB myWindow)
Operation

Returns the height in pixels of the caption area for the specified dialog box.

Example

DB DBox = create("Dialog Box", styleCentered|styleFixed)
int i = getCaptionHeight(DBox)
label(DBox, "Caption height is " i "")
show DBox

help, gluedHelp

Declaration

void {gluedH|h}elp(DB box,
    int index)

Operation

Adds a Help button to a dialog box box. When the user clicks the button, help is activated displaying the entry identified by the index number.

The optional second argument associates the Help button with the named helpFile and an entry index in it. The help file must be in the appropriate format for the platform and must be referenced by a full path name; a relative path does not work in this case. This can be used to add user-defined help information to Rational DOORS.

When a dialog box has a large number of buttons, the gluedHelp function is used to link the help button to the last button, to prevent them from overlapping.

These functions can only refer to help entries in the standard Rational DOORS help file, DOORS.HLP. In addition the following standard values can be used to obtain help system functions:

1   Contents page
2   Help on help
3   Search help

Example

help(simParse, 301)
help(simParse, "SIMPARSE.HLP", 1)

hide(dialog box)

Declaration

void hide(DB box)
Operation
Removes dialog box box from the screen.

Example
hide thisBox

---

**raise**

**Declaration**
void raise(DB box)

**Operation**
Brings dialog box box to the top, over all other windows.

**Example**
raise tempBox

---

**setFocus**

**Declaration**
void setFocus(Module m)

**Operation**
Sets the windows focus on the module m.

---

**ready**

**Declaration**
void ready(DB box)

**Operation**
Used after a call to busy, this function makes dialog box box sensitive to input again, and removes the waiting cursor.

**Example**
ready graphBox

---

**realize(pending)**

**Declaration**
void realize(DB box)
Operation
Creates and displays the dialog box without suspending execution of the DXL program. The dialog box only becomes active when a \texttt{show} function is called, either for this dialog box or another.

This function is used where you wish to do something that can only be done once the dialog box internal structure has been created, for example, add columns to a list view. Creating the internal structure is called \texttt{realization}.

Example
\begin{verbatim}
realize infoBox
\end{verbatim}

\underline{realize(show)}

\underline{Declaration}
\begin{verbatim}
void realize(DB myWindow,
         int x,
         int y)
\end{verbatim}

\underline{Operation}
Creates the specified window and initializes its origin to the co-ordinates \((x,y)\).

\underline{release}

\underline{Declaration}
\begin{verbatim}
void release(DB modalBox)
\end{verbatim}

\underline{Operation}
Hides the modal dialog box \texttt{modalBox}, and resumes execution of the DXL program after the call to \texttt{block}. The Rational DOORS interface then becomes operative.

Example
\begin{verbatim}
release importantQuesBox
\end{verbatim}

\underline{show(dialog box)}

\underline{Declaration}
\begin{verbatim}
void show(DB box)
\end{verbatim}

\underline{Operation}
Displays the dialog box and suspends execution of the DXL program. Execution only continues in callbacks from the buttons on the dialog box. No code should appear after a \texttt{show} as it would never be executed.

Example
\begin{verbatim}
show splashBox
\end{verbatim}
**showing**

*Declaration*

```c
bool showing(DB box)
```

*Operation*

Returns true if `box` is displayed as a result of a call to `show` or `realize`.

*Example*

```c
if (showing infoBox) { ... }
```

**getParent**

*Declaration*

```c
{DB|DBE} getParent(DBE element)
```

*Operation*

Returns the parent dialog box or dialog box element of the specified dialog box element. This is useful in callback functions.

If the function that returns an object of type DBE is called, and the parent is not an object of type DBE, the function returns null.

*Example*

```c
void takeAction(DBE button) {
    DB enclosedby = getParent button
    // user code here
} // takeAction
```

**setParent**

*Declaration*

```c
void setParent(DB box|DBE child, {DB|DBE|Module} parent)
```

*Operation*

Sets the parent of `child` to be `parent`.

The only type of DBE which can be the parent of another DBE, is a frame.
setPos

Declaration

```c
void setPos(DB myWindow,
            int x,
            int y)
```

Operation

Sets the screen co-ordinates of the origin of the specified window to the co-ordinates \((x, y)\).

setCenteredSize

Declaration

```c
void setCenteredSize(DB box,
                      int width,
                      int height)
```

Operation

Sets the width and height of `box` to `width` and `height` pixels, independently of any styles used, such as `styleCentered` or `styleFixed`.

This function must be placed after a call to the `realize(pending)` function, and before any further call to either the `show(dialog box)` or `block` functions.

Example

```c
DB dlg = create("Test Window", styleCentered |
                styleFixed)
realize dlg
// both width and height are specified in pixels
setCenteredSize(dlg, 300, 100)
show dlg
```

setSize

Declaration

```c
void setSize(DB myWindow,
             int w,
             int h)
```

Operation

Sets the width and height of the specified window to the values in `w` and `h`. Dimensions are specified in pixels.
setTitle

Declaration
void setTitle(DB myWindow,
                    string newTitle)

Operation
Sets the title of the specified window to newTitle. This function is used after the window is created.

setBaseWindowContext

Declaration
void setBaseWindowContext()

Operation
Used when displaying dialog boxes in batch mode. This enables the use of realize() for populating DBEs.

startConfiguringMenus

Declaration
void startConfiguringMenus({DB box|DBE element})

Operation
Starts menu creation and configuration in box or element. To stop menu creation and configuration for a dialog box element, use the stopConfiguringMenus function. For a dialog box, the menu configuration stops when the dialog box is shown.

stopConfiguringMenus

Declaration
string stopConfiguringMenus(DBE element)

Operation
Disables menu creation and configuration functions for the specified dialog box element. To start menu creation and configuration, use the startConfiguringMenus function.
topMost

Declaration
DB topMost(string title)

Operation
Creates a dialog box that always stays on top of all other windows. This can be used instead of the create(dialog box) function.

Example
DB top = topMost "TOPMOST"
label(top, "I am on top!")
show top

hasFocus

Declaration
bool hasFocus(DBE toolbar)

Operation
Returns true if the supplied toolbar DBE contains an element that currently has the keyboard focus. Otherwise, returns false.

setDXLWindowAsParent

Declaration
void setDXLWindowAsParent(DB dialog)

Operation
Sets the DXL interaction window to be the parent of dialog. If there is no DXL interaction window, the parent is set to null.

minimumSize

Declaration
void minimumSize(DB box, int width, int height)

Operation
Sets the minimum size of the dialog box to be width and height.
Negative values are ignored. A width value of 200 and a height value of -1 only sets the minimum width for the dialog box. Note the dialog box does not need to be realized to set the minimum size.

**Example**

```dxl
DB db = create "hello"
realize db
minimumSize(db, 400, 100)
show db
```

---

## Dialog box elements

Dialog box elements define the components of a dialog box. These are called **controls** on Windows, and **Widgets** on Motif, the most common user interface tool kit on UNIX.

Dialog box elements provide a wide range of capability, although all have the **DBE** data type. This manual groups the functions for DXL dialog box elements into the following categories:

- Common element operations
- Simple elements for dialog boxes
- Choice dialog box elements
- View elements
- Text editor elements
- Buttons
- Canvases
- Complex canvases

---

## Common element operations

This section defines element operations. Unless otherwise specified, these functions can be used with all dialog box elements.

For dialog box elements, the **set** function has many different variants, all of which are defined in this section. There are pointers to the appropriate **set** function from other sections within this chapter.
### addMenu

#### Declaration

```dxl
void addMenu(DBE element,  
    string title,  
    char mnemonic,  
    string entries[],  
    char mnemonics[],  
    char hots[],  
    string help[],  
    string inactiveHelp[],  
    [, int noOfEntries,]  
    Sensitivity sensitive(int entryIndex),  
    void callback(int entryIndex))
```

#### Operation

Adds a menu to a menu bar, canvas, list view, or tree view. If `element` is a menu bar, the new menu appears after any other menus. If `element` is a canvas, list view, or tree view, the new menu is activated by a right click. For further information on creating the dialog box elements that can take menus, see the `menuBar`, `canvas`, `listView`, and `treeView` functions.

The arguments passed are divided into two sets: those that define the menu, and those that define the menu entries, which are specified as arrays. To use fixed-size arrays all containing the same number of elements, omit `noOfEntries`. To use freely-defined arrays, specify the minimum number of elements in `noOfEntries`.

The arguments passed to the function are defined as follows:

- **element** - The menu bar or canvas in which the menu is to appear; this is returned by a call to the `menuBar` or `canvas` function.
- **title** - The title of the menu, as it appears in the menu bar.
- **mnemonic** - The keyboard access character, normally shown underlined, which activates the menu when pressed with ALT; the value `ddbNone` means that there is no mnemonic.
- **entries** - The strings that appear in the menu.
- **mnemonics** - The keyboard access character for this option, normally shown underlined, which activates the option when pressed with ALT; the value `ddbNone` indicates that there is no mnemonic.
- **hots** - A hot key that directly activates the option when pressed with CTRL; for example, if the value of `hots[3]` is S, CTRL+S activates the third option of the menu; the value `ddbNone` indicates that there is no hot key.
You can construct one level of cascading menus by placing a right angle bracket (>) character at the start of an option name, indicating that it is a member of a sub-menu:

```cpp
const string formatMenu[] = {
    "Size",
    ">Small",
    ">Normal",
    ">Large",
    "Style",
    ">Bold",
    ">Italic"
};
```

This constructs a cascading menu. The first cascading menu, Size, opens out, followed by the second cascading menu, Style.

Finally, two callback functions are required: one to determine whether menu items are sensitive, and one that is called when a menu option is activated.

The function `sensitive(int entryIndex)` is called for each option, each time the menu is displayed. The function must return one of the following values:

<table>
<thead>
<tr>
<th>Availability</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddbUnavailable</td>
<td>The menu option is grayed out.</td>
</tr>
<tr>
<td>ddbAvailable</td>
<td>The menu option is active.</td>
</tr>
<tr>
<td>ddbChecked</td>
<td>The menu entry is active and has a check beside it.</td>
</tr>
</tbody>
</table>

When the user selects an option, `callback(int entryIndex)` is called with the index of the option, and your program must perform the appropriate operation. For both `sensitive` and `callback` functions, `entryIndex` starts at 0, and counts up, including cascading menu entries, so there is a direct correspondence between the array elements and the index returned by the menu.

---

### active

**Declaration**

```cpp
void active(DBE element)
```

**Operation**

Sets an item active, restoring it from being grayed out and enabling users to interact with it. This is the opposite of the `inactive` function. The `active` function can be used with any kind of dialog box element.
Example
if (gotFileName) active startLoader

inactive

Declaration
void inactive(DBE)

Operation
Sets an item inactive, displaying it in gray and preventing users from interacting with it. This is the opposite of the active function. The inactive function can be used with any kind of dialog box element.

Example
if (dataNotComplete) inactive verify

hide

Declaration
void hide(DBE element)

Operation
Hides a single dialog box element.

Example
hide showAdminButtons

setGotFocus

Declaration
void setGotFocus(DBE element, void callback(DBE element))

Operation
Sets the callback function to call when element gets input focus. Currently, element must be a list view or tree view on a Windows platform.

setLostFocus

Declaration
void setLostFocus(DBE element, void callback(DBE element))
Operation
Sets the callback function to call when \texttt{element} loses input focus. Currently, \texttt{element} must be a list view or tree view on a Windows platform.

\section*{show(\texttt{element})}

\textbf{Declaration}

\begin{verbatim}
void show(DBE element)
\end{verbatim}

\textbf{Operation}

Makes a single dialog box element visible again.

\textbf{Example}

\begin{verbatim}
show showAdminButtons
\end{verbatim}

\section*{delete(\texttt{option or item})}

\textbf{Declaration}

\begin{verbatim}
void delete(DBE element, int index)
\end{verbatim}

\textbf{Operation}

Deletes the option in \texttt{element} at the given \texttt{index}. The argument \texttt{element} can be a choice, tab strip, list, multi-list, combo box, or list view. Positions start at zero; when an element is deleted, all the others are moved down. The last element cannot be deleted in a tab strip. To delete all items in a list or list view, use the \texttt{empty} function.

\textbf{Example}

\begin{verbatim}
delete(components, obsoleteEntry)
\end{verbatim}

\section*{delete(\texttt{item in tree view})}

\textbf{Declaration}

\begin{verbatim}
void delete(DBE treeView, string path)
\end{verbatim}

\textbf{Operation}

Deletes the item pointed to by \texttt{path}, which must be an absolute path.

\textbf{Example}

\begin{verbatim}
delete(treeView, "Project/Module1")
\end{verbatim}
empty

**Declaration**

```c
void empty(DBE element)
```

**Operation**

Deletes all items in a list, multi-list, choice, combo box, list view or tree view.

**Example**

```c
empty listView1
```

insert(option or item)

**Declaration**

```c
void insert(DBE element, int index, string value)
```

**Operation**

Inserts a new value into element at position index. The argument element can be a choice, tab strip, list, multi-list, combo box, or list view. Positions start at zero; when a new element is inserted all the other values are moved up. This function inserts duplicate values if they are specified.

**Example**

```c
insert(months, 4, "May")
```

insert(item in list view)

**Declaration**

```c
void insert(DBE listView, int row, string value, Icon icon)
```

**Operation**

Inserts a new item with the specified string value into the list view, at the zero based row number. The icon is the icon that appears to the left of the string value on the specified row.

insert(item in tree view)

**Declaration**

```c
void insert(DBE treeView, string path, Icon normal, Icon selected)
```
Operation

Inserts the item pointed to by path into treeView. The third and fourth arguments define icons for the item when it is not selected and selected, respectively. To make the selected icon the same as the normal icon, use iconNone as the value for selected. For valid icon values, see “Icons,” on page 483.

Note that the slash character has a special meaning when included in a string to be inserted: it represents a parent-child relationship. So adding “Heading1” then “Heading1/sub1” will add “Heading1” as a top-level entry, and “sub1” as a child entry under it.

Example

insert(treeView, newFolder, iconFolder, iconFolderOpen)

---

**noElems**

**Declaration**

```plaintext
int noElems(DBE element)
```

**Operation**

Returns the number of options or items in element. The argument element can be a choice, tab strip, list, multi-list, combo box, or list view.

**Example**

```plaintext
int noOfResources = noElems resourceList
string listContents[noOfResources]
int i
for (i = 0; i < noOfResources; i++)
    listContents = get(resourceList, i)
```

---

**select(element)**

**Declaration**

```plaintext
void select(DBE textElement, int start, int end)
```

**Operation**

Selects text only in a rich text or rich field dialog box element.

---

**selected(element)**

**Declaration**

```plaintext
bool selected(DBE element, int index)
```
Operation

Returns `true` if the option or item identified by `index` is selected; otherwise returns `false`. The argument `element` can be a list, multi-list, or list view.

Example

```dxl
if (selected(products, ownBrand))
    print "Using own brand\n"
```

### selected(item)

**Declaration**

```dxl
def selected(DBE treeView, string path)
```

**Operation**

Returns `true` if the item pointed to by `path` is selected; otherwise returns `false`. The argument `path` must be an absolute path.

### get(element or option)

**Declaration**

```dxl
def get(DBE element [,int index])
```

**Operation**

For a multi-list element, returns a value for the most recently selected/de-selected item. For all other elements, with one argument, returns a value for the first or only selected element of the appropriate type. The optional second argument is available only for a string return type and list views or choice dialog box elements. Use it to specify a given item in a list view or a given position in a choice element. The return types and values for all dialog box elements are as follows:

<table>
<thead>
<tr>
<th>Element</th>
<th>Return type</th>
<th>Contents of most recently selected/deselected option</th>
<th>Return value if no selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>canvas</td>
<td>not supported</td>
<td>integer defining which element or elements are checked; when converted to binary, the value is a bitmap for the selection of check boxes, for example, 5 (101) means first and third boxes checked</td>
<td>0</td>
</tr>
<tr>
<td>check box</td>
<td>int</td>
<td>contents of selection (chosen or typed) or contents of specified choice</td>
<td>null string</td>
</tr>
<tr>
<td>choice</td>
<td>string</td>
<td>contents of selection (chosen or typed) or contents of specified choice</td>
<td>null string</td>
</tr>
<tr>
<td>Element</td>
<td>Return type</td>
<td>Contents of most recently selected/deselected option</td>
<td>Return value if no selection</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>------------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>int</td>
<td></td>
<td>index (position) of selected option except for typed entries, which return -1 even if the typed entry matches a selection. This is the preferred method when the value being retrieved is to be used elsewhere. The number should be used as the index to retrieve the value from the original string array.</td>
<td>-1</td>
</tr>
<tr>
<td>field</td>
<td>string</td>
<td>contents of field</td>
<td>null string</td>
</tr>
<tr>
<td>bool</td>
<td></td>
<td>if the DBE is read only, returns true; otherwise, returns false</td>
<td></td>
</tr>
<tr>
<td>file name</td>
<td>string</td>
<td>path in file selector</td>
<td>null string</td>
</tr>
<tr>
<td>frame</td>
<td>not supported</td>
<td></td>
<td></td>
</tr>
<tr>
<td>list</td>
<td>string</td>
<td>contents of selected option or specified option</td>
<td>null string</td>
</tr>
<tr>
<td>list view</td>
<td>string</td>
<td>value of selected item or specified item</td>
<td>null string</td>
</tr>
<tr>
<td>multi-list</td>
<td>string</td>
<td>contents of first selected option or specified option</td>
<td>null string</td>
</tr>
<tr>
<td>radio box</td>
<td>int</td>
<td>index of the selected option in the array</td>
<td>not applicable</td>
</tr>
<tr>
<td>rich field</td>
<td>string</td>
<td>contents of rich field</td>
<td>null string</td>
</tr>
<tr>
<td>rich text</td>
<td>string</td>
<td>contents of rich text box</td>
<td>null string</td>
</tr>
<tr>
<td>slider</td>
<td>int</td>
<td>integer in range specified</td>
<td>-1</td>
</tr>
<tr>
<td>tab strip</td>
<td>string</td>
<td>name</td>
<td>not applicable</td>
</tr>
<tr>
<td></td>
<td>int</td>
<td>index (position) of currently selected tab</td>
<td>not applicable</td>
</tr>
</tbody>
</table>
## get

You can find out the read-only status of a text or string DBE using `get` in a boolean expression.

**Example**

```dxl
DB exBox = create "Use of Get"
DBE intIn = slider(exBox, "Integer:", 50, 0, 100)
DBE stringIn = field(exBox, "String:", "Example", 20)

void doGet(DB exBox) {
    int i = get intIn
    string s = get stringIn
    print i " , " s "\n"
}

apply(exBox, "Get", doGet)
show exBox
```

### setTextChangeCB

**Declaration**

```dxl
void setTextChangeCB(DBE, void callbackFn (DBE))
```

Sets the text change callback for field, richField, text, richText DBEs where the callback is of the form `void callbackFn(DBE)`.

When the callback function is invoked on a text change, DBE will be the handle of the edit control DBE, which can be field, richField, text or richText.

Calling this perm using any type of DBE other than these four, leads to a run-time DXL error.

**Example**

```dxl
DB db
```
db = create("hello world", styleCentered | styleStandard)
DBE dbe = field(db, "label", ",", 34, false)
int count = 0
void cb(DBE dbe)
{
    count++
    string str = plainText(get(dbe))
    print count "). The edit text now is \"" str \"\".
}
setTextChangeCB(dbe, cb)
void checkText(DB db)
{
    hide db
}
ok(db, "Cancel", checkText)
close(db, false, checkText)
show db

**toolBarEditGetString**

**Declaration**

*string* toolBarEditGetString(DBE *tb*, int *index*)

Gets the contents of the edit control hosted on the toolbar with DBE handle *tb*, where *index* identifies the edit control on this toolbar by the index of the edit control.

**Example**

```dxml
#include "utils/icons.inc"
DB db = create("hello", styleStandard)
ToolType types[] = {toolEditField, toolToggle, toolToggle, toolToggle, toolToggle, toolToggle, toolToggle}
Sensitivity mappingCallback()
{
    return ddbAvailable
}
```
bool toggleStates[6] = {false, true, true, true, true, true}
int param[] = {100, ID_CSAVE, ID_CPRINT, ID_CTEXTBOLD, ID_CTEXTBULLET, ID_CTEXTITALIC}
string toolTip[] = {"first", "second", "third", "fourth", "fifth", "six"}
string help[] = {"firsthelp", "secondhelp", "thirdhelp", "comboonehelp", "fourthhelp", "combotwohelp"}
string inactiveHelp[] = {"firstinchelp", "secondinchelp", "thirdinchelp", "combooneinchelp", "fourthinhelp", "combotwoinchelp"}

DBE tb = null
Sensitivity sensitive(int entryIndex)
{
    if (types[entryIndex] == toolEditField)
    {
        return ddbAvailable
    }
    if (null toolBarEditGetString(tb, 0))
    {
        return ddbUnavailable
    }
    if (toggleStates[entryIndex])
    {
        return ddbChecked
    }
    return ddbAvailable
}
int iCount = 0
void callback(int entryIndex)
{
    if (types[entryIndex] == toolEditField)
    {
        iCount++
        print iCount "). String in edit box with index " entryIndex " is "," toolBarEditGetString(tb, entryIndex) ".
        if (null toolBarEditGetString(tb, 0))
{
    toggleStates[1] = true
    toggleStates[2] = true
    toggleStates[3] = true
    toggleStates[4] = true
    toggleStates[5] = true
}
updateToolBars(db)
}
else if (types[entryIndex] == toolToggle)
{
    toggleStates[entryIndex] = (!toggleStates[entryIndex])
}
}  
tb = toolBar(db,
"tbname",
mappingCallback,
types,
param,
toolTip,
help,
inactiveHelp,
6,
sensitive,
callback,
true,
true)
realize db
setSize(db, 500, 500)
updateToolBars(db)
show db
get(selected text)

Declaration

```cpp
bool get(DBE textElement, int &first, int &last)
```

Applies only to text dialog box elements. It returns `true` if there is a selected area of text; otherwise, returns `false`. If it returns `true`, the integers return the start and finish indices of the selected text, starting from 0. For example, if the first ten characters are selected, `first` and `last` contain 0 and 9.

set(value or selection)

Declaration

```cpp
void set(DBE element, {string|int|bool} value)
void set(DBE currDBE, Buffer b)
```

Operation

The first form sets either the value of an element or the status of the selected element as follows:

The second form sets the content of the specified DBE to be the content of the Buffer:

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>canvas</td>
<td>not supported</td>
<td></td>
</tr>
<tr>
<td>choice</td>
<td>int</td>
<td>Sets the selected option.</td>
</tr>
<tr>
<td>check box</td>
<td>int</td>
<td>Sets the selected option.</td>
</tr>
<tr>
<td>field</td>
<td>string</td>
<td>Sets the contents.</td>
</tr>
<tr>
<td></td>
<td>bool</td>
<td>When <code>true</code>, sets field read only; otherwise, sets field read/write.</td>
</tr>
<tr>
<td>file name</td>
<td>string</td>
<td>Sets the contents.</td>
</tr>
<tr>
<td>frame</td>
<td>string</td>
<td>Sets the contents.</td>
</tr>
<tr>
<td>label</td>
<td>string</td>
<td>Sets the contents.</td>
</tr>
<tr>
<td>list</td>
<td>int</td>
<td>Sets the selected option.</td>
</tr>
<tr>
<td>list view</td>
<td>int</td>
<td>Sets the selected item.</td>
</tr>
<tr>
<td>multi-list</td>
<td>int</td>
<td>Sets the selected option.</td>
</tr>
<tr>
<td>text</td>
<td>string</td>
<td>Sets the contents.</td>
</tr>
<tr>
<td></td>
<td>bool</td>
<td>When <code>true</code>, sets text read only; otherwise, sets text read/write.</td>
</tr>
<tr>
<td>radio box</td>
<td>int</td>
<td>Sets the selected item.</td>
</tr>
</tbody>
</table>
Using set with -1 deselects any selection in a list, choice or radio button dialog box element.

If these functions are used with an incorrect type DBE, a DXL run-time error occurs.

**Example**

```dxl
DB exBox = create "Use of Put"

DBE intOut = slider(exBox, "Integer:", 50, 0, 100)

DBE stringOut = field(exBox, "String:", "Example", 20)

void doHigh(DB exBox) {
    set(intOut, 100)
    set(stringOut, "Max out")
} // doHigh

void doLow(DB exBox) {
    set(intOut, 0)
    set(stringOut, ")")
} // doLow

apply(exBox, "Low", doLow)
apply(exBox, "High", doHigh)
show exBox
```
set(selected status)

Declaration

void set(DBE list, int index, bool selected)

Operation

Sets the status of a selected item within a list or list view, identified by index in a list or list view. Valid items are ranged between position 0 and a number that can be obtained from:

noElems(DBE)-1

If this function is used with an incorrect type DBE, a DXL run-time error occurs.

set(choice element values)

Declaration

void set(DBE choice, string choices[ ] [,int noOfChoices])

Operation

Sets a new range of values into a choice element. You can supply a complete array of strings or a partially filled array with the number of items supplied in the noOfchoices argument.

This works only with choice dialog box elements created with the choice function. If this function is used with an incorrect type DBE, a DXL run-time error occurs.

Example

string attrNames[100]
int noOfAttrs = 0
string an
for an in current Module do
  attrNames[noOfAttrs++] = an
set(attrChoice, attrNames, noOfAttrs)

set(item value)

Declaration

void set(DBE listView, int item, int column, string value)

Operation

Sets the value of a specific column item within a list view.

If this function is used with an incorrect type DBE, a DXL run-time error occurs.
set(status bar message)

Declaration

void set(DBE statusBar, int section, string message)

Operation

Sets the value of a particular section within a status bar.

If you use ddbFullStatus as section, the string is displayed in the full width of the status bar, as with menu help. To return to normal display, specify ddbFullStatus with a null string for message.

If this function is used with an incorrect type DBE, a DXL run-time error occurs.

set(file selector)

Declaration

void set(DBE fileSelector, string descs, string exts)

Operation

Sets the file selector description(s) and extension(s) for a dialog box file selector.

If this function is used with an incorrect type DBE, a DXL run-time error occurs.

Example

DB b = create "File Selector DB"
DBE fs = fileName b
set(fs, "Comma separated files", ".CSV")
show b

set(icon)

Declaration

void set(DBE element, int index, [int column,] Icon icon)

Operation

Sets the displayed icon for either a tab in a tab strip or item in a list view that is identified by index to have the specified icon. The column argument must be passed for list views, but not for tab strips. For possible values of icon, see “Icons,” on page 483. Use this function with iconNone as the value for icon to remove an icon.

If this function is used with an incorrect type DBE, a DXL run-time error occurs.

Example

set(linkList, 2, iconLink)
set(select)

Declaration

```c
void set(DBE element, void select(DBE))
```

Operation

Attaches a callback to any dialog box element other than a list view. The callback must be of the form:

```c
void select(DBE option){
}
```

which fires when `option` changes.

The exact semantics vary depending on the type of element, but in principle it means a single click. For field elements, the callback only fires when the user clicks **Return** or **Enter** with the cursor in the field.

If this function is used with a list view, a DXL run-time error occurs.

Example

This example adds a callback to a radio box.

```c
DB boatBox = create "Craft"
string boats[] = {"Dinghy", "Destroyer", "Carrier", "Mine sweeper"}
DBE boatCheck = radioBox(boatBox, "Select class:", boats, 3)
void toBuild(DBE option) {
    int favorite = get option
    ack(boatBox, "You are planning a new " boats[favorite] "?")
} // toBuild
set(boatCheck, toBuild)
show boatBox
```

set(key or mouse callback)

Declaration

```c
void set(DBE canvas, void callback(DBE canv,
       {char key|int button}
       bool controlDown,
       int x,
       int y})
```
Operation

Attaches a callback to the specified canvas. The callback can be fired from character input or a mouse click, depending on the second argument passed to the callback function.

For a character input callback you must supply the code for the key, whether the control key was down, and the mouse position when the key was pressed. The key code is normally the ASCII character value, but might be one of a set of predefined constants (see “Keyboard event constants,” on page 560).

For a mouse click callback you must supply the canvas identifier, the mouse button number, starting from 1 for the left button, whether the control key was down, and the co-ordinates of the mouse at the time.

If this function is used with an incorrect type DBE, a DXL run-time error occurs.

Example

This example adds a callback to a canvas.

```dxl
// character input callback
DB typeBox = create "Type Something"
int col = 0
void redraw(DBE x) {
    draw(x, 20, 20, "Type something!")
} // redraw
DBE can = canvas(typeBox, 300, 300, redraw)
void key(DBE can, char k, bool ctrl, int x, int y) {
    color(can, col)
    if (k == keyF2) {
        col++
    } else if (k == keyF3) {
        background(can, col++)
    } else {
        draw(can, x, y, k "")
    }
    if (col > 29) col = 0
} // key
set(can, key)
show typeBox
// mouse button callback
DB drawBox = create "Test"
void redraw(DBE x) {
    draw(x, 20, 20, "Hello!")
} // redraw
DBE can = canvas(drawBox, 300, 300, redraw)
int lastX = -1
```
int lastY = -1
int firstX
int firstY
int col = 0

void btn(DBE can, int bt, bool ctrl, int x,
int y) {
if (bt == 1) {
if (lastX > 0) {
    line(can, lastX, lastY, x, y)
} else {
    rectangle(can, x, y, 1, 1)
    firstX = x
    firstY = y
}
    lastX = x
    lastY = y
} else if (bt == 2) {
    lastX = -1
} else if (bt == 3) {
    col++
    if (col > 29) col = 0
    color(can, col)
}
} // btn
set(can, btn)
show drawBox

set(select and activate)

Declaration

void set(DBE element, void select(DBE), void activate(DBE))

Operation

Attaches two callback functions to a list or tree view.
The first callback fires when an item is selected (a single click); the second fires when an item is activated (a double click).
Both callbacks must be of the form:
void callback(DBE item){
}
If this function is used with an incorrect type DBE, a DXL run-time error occurs.
Example

```dxl
DB listBox = create "The Good Numbers"
string states[]={"New Jersey", "Virginia", "Texas", "California", "Europe"}
string phones[]={"201 442-4600", "703 904-4360", "817 588-3008", "408 879-2344", 
"+44 1865 784285"}
DBE abcList = list(listBox, "ABC Offices:", 200, 4, states)
full listBox
DBE telNo = field(listBox, "Telephone:", ",", 30, true)
void onSelection(DBE l) {
    int sel = get abcList
    if (sel >= 0) {
        set(telNo, phones[sel])
    } else {
        set(telNo, ",")
    }
} // onSelection
void onActivate(DBE l) {
    int sel = get abcList
    if (sel >= 0) {
        ack(listBox,
            "Calling ABC in " states[sel] " on 
            phones[sel])
    }
} // onActivate
set(abcList, onSelection, onActivate)
show listBox
```

set(list view callback)

Declaration

```dxl
void set(DBE listView, void callback(DBE, int))
```

Operation

Attaches a callback to a check box within a list view, provided the list view was created with check boxes (using the `listViewOptionCheckboxes` style). The callback must be of the form:

```dxl
void select(DBE listView, int selected){
}
```

which fires when the state of any check box changes. The `selected` argument identifies the item that changed.

If this function is used with an incorrect type `DBE`, a DXL run-time error occurs.
set(select, deselect, and activate)

Declaration

```c
void set(DBE listView,
          void select(DBE, int),
          void deselect(DBE, int),
          void activate(DBE, int))
```

Operation

Attaches three callback functions to a list view.

The first callback fires when an option is selected (a single click); the second fires when an option is deselected (a side effect of a single click on another item); the third fires when an item is activated (a double click).

All callbacks must be of the form:

```c
void select(DBE listView, int selected){
}
```

If this function is used with an incorrect type DBE, a DXL run-time error occurs.

set(sort function)

Declaration

```c
void set(DBE listView,
          int columnIndex,
          int dxlSortFn(string, string))
```

Operation

Attaches a sort function to a specific column within a list view. The callback must be of the form:

```c
void dxlSortFn(string s1, string s2){
}
```

The sort function must return the following values:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1==s2</td>
<td>0</td>
</tr>
<tr>
<td>s1&gt;s2</td>
<td>1</td>
</tr>
<tr>
<td>s1&lt;s2</td>
<td>-1</td>
</tr>
</tbody>
</table>

If this function is used with an incorrect type DBE, a DXL run-time error occurs.
set(tree view expand)

**Declaration**

```cpp
void set(DBE treeView, bool expand(DBE, string))
```

**Operation**

Attaches a callback to a tree view. The callback fires when an attempt is made to expand a specific branch. The callback must be of the form:

```cpp
void expand(DBE treeView, string branch){
}
```

The callback function must return the following values:

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow expansion</td>
<td>true</td>
</tr>
<tr>
<td>Refuse expansion</td>
<td>false</td>
</tr>
</tbody>
</table>

If this function is used with an incorrect type `DBE`, a DXL run-time error occurs.

**setFocus**

**Declaration**

```cpp
void setFocus(DBE element)
```

**Operation**

Sets keyboard focus to the specified element.

**getBuffer(DBE)**

**Declaration**

```cpp
Buffer getBuffer(DBE element)
```

**Operation**

Creates a new buffer object and returns it. The returned buffer contains the value of the specified `DBE` element.

**Example**

```cpp
DB exBox = create "DBE example"
DBE stringIn = field(exBox, "String:", "Example", 20)
void doGet(DB exBox) {
    Buffer b = create
```
b = getBuffer(stringIn)
    print b "\n"
} // doGet
apply(exBox, "Get", doGet)
show exBox

setFromBuffer(DBE, Buffer)

Declaration
void setFromBuffer(DBE element, Buffer b)

Operation
Sets the contents of the specified DBE element from the contents of the specified buffer b.

Example
DB exBox = create "DBE example"
DBE stringIn = field(exBox, "String:", "Example", 20)
Buffer b = create
b = "test setting DBE from buffer"
setFromBuffer(stringIn, b)
show exBox

useRTFColour

Declaration
void useRTFColour(DBE dbe, bool useRTF)

Operation
If dbe is a rich text box or a rich text field, then:
• If useRTF is true, the underlying editbox will use the rtf color markup instead of the default color for text in dialog boxes
• If useRTF is false, the underlying editbox will use the default system color for text in dialog boxes
• If the dbe is not rich text or a rich text field, nothing happens
• If the dbe has not been realized, nothing happens

Example
DB test = create("Test text db")
DBE textdbe = richText(test, "test", "initial", 200, 200, false)
string colourstring = "\{\rtf1\ansi\ansicpg1252\deff0\deflang1033\fonttbl\f0\fswiss\fcharset0 Arial;\}\{\colortbl ;\red255\green0\blue0;\red255\green0\blue255;\red0\green0\blue128;\}\viewkind4\uc1\pard\f0\fs20 Some\cf1 text \cf0 with \cf2 different\cf0 \cf3 colors\cf0 in it.\par \par\}" 

realize test
useRTFColour(textdbe, true)
set(textdbe, colourstring)
show test

Simple elements for dialog boxes

This section defines functions for simple elements such as two-state options, with the exception of buttons, which are defined in “Buttons,” on page 556. More complex elements that allow the user to choose from various options are defined in “Choice dialog box elements,” on page 540.

label

Declaration
DBE label(DB box, string label)

Operation
Creates a label element in dialog box box.

Example
DB infoBox = create "About SimParse"
label(infoBox, "SimParse V2.1")
show infoBox
separator(dialog box)

Declaration

DBE separator(DB box)

Operation

Places a full width separating line across dialog box box.

Example

This example creates a separator between the input slider and the output field. Dialog boxes normally include a separator, which is automatically created, between the user-defined elements and the standard buttons.

DB exBox = create "Use of Separator"
DBE input = slider(exBox, "Input:", 50, 0, 100)
separator exBox
DBE output = field(exBox, "Output:", "", 30)
void calc(DB exBox) {
    int i = get input
    set(output, "Input was " i ")
} // calc
apply(exBox, calc)
show exBox

splitter

Declaration

DBE splitter(DB box,
            DBE left,
            DBE right,
            int width)

Operation

Places a movable vertical separating line across dialog box box. The arguments define the left part of the dialog box, the right part of the dialog box, and the width of the splitter in pixels. This is only supported for DBEs of type listView or treeView.

Example

// constants
const string SARR_DUMMY[ ] = {}
const int TREE_HEIGHT = 10
const int LIST_WIDTH = 300
const int LIST_HEIGHT = 10

// dxl dialogs
DB dlg = null

// dxl elements
DBE dbeTree, dbeList, dbeSplitter

// create dialog
dlg = create("Test", styleCentered)

// tree
dbeTree = treeView(dlg, 0, TREE_WIDTH,
    TREE_HEIGHT)
dbeTree->"top"->"form"
dbeTree->"left"->"form"
dbeTree->"bottom"->"form"
dbeTree->"right"->"unattached"

// list
dbeList = listView(dlg, 0, LIST_WIDTH,
    LIST_HEIGHT, SARR_DUMMY)
dbeList->"top"->"aligned"->dbeTree
dbeList->"left"->"unattached"
dbeList->"bottom"->"form"
dbeList->"right"->"form"

// splitter
dbeSplitter = splitter(dlg, dbeTree, dbeList, 4)
dbeSplitter->"top"->"form"
dbeSplitter->"left"->"unattached"
dbeSplitter->"bottom"->"form"
dbeSplitter->"right"->"unattached"

realize dlg
{
    // information is displayed over a single
    // column
    insertColumn(dbeList, 0, "Name", LIST_WIDTH -
        20, null)
}
Declaration

DBE frame(DB box,
    string label
    [,int width,
      int height])

Operation

Creates a frame element in box, which can contain other elements. The label is the title of the frame; width and height specify the size of the frame in pixels. If width and height are omitted, the frame expands to fit the elements within it.

Example

This example creates a tab strip and frame, and places the frame inside a tab.

const string tabStrings[] = {"A", "B", "C"}
DB box = centered "Example"
DBE theTab
DBE theFrame

void tabCb(DBE xx) {
    int i = get xx
    if (i == 0) {
        show theFrame
    } else {
        hide theFrame
    }
}

theTab = tab(box, tabStrings, 300, 300, tabCb)
// attach all the edges of the tabstrip to the // form
theTab->"left"->"form"
theTab->"right"->"form"
theTab->"top"->"form"
theTab->"bottom"->"form"
theFrame = frame(box, "A frame", 100, 100)
// place the frame inside the tabstrip
theFrame->"left"->"inside"->theTab
theFrame->"right"->"inside"->theTab
theFrame->"top"->"inside"->theTab
theFrame->"bottom"->"inside"->theTab
realize box
// ensure widgets are showing for correct tab
tabCb theTab
show box

fileName

Declaration

DBE fileName(DB box,
    [string label,]
    [,string initFileName
    [,string extension,
        string description
    [,bool readOnly]]])

Operation

Creates a window-wide element inside the specified dialog box for capturing a file name. As in other Rational DOORS windows, there is a field for the file name and a button, Browse, to invoke a file selector window. Optionally, the element is called label.

When present, the initFileName argument provides an initial value, which can be an absolute or relative path.

The fourth and fifth optional arguments allow you to specify a file extension and description, which fill the File of type box. Note that not all platforms make use of this additional information.

When the readOnly argument is true, it checks the Open as read-only box. Note that not all platforms make use of this additional information.

Example

// basic file name
DBE fn = fileName(loader, "input.dat")
// file spec and description added
DBE fn = fileName(load, "input.dat", ".dat", "Data files")
### field

**Declaration**

```
DBE field(DB box,  
    string label,  
    string initial,  
    int width  
    [,bool readOnly])
```

**Operation**

Creates a single-line text-field element. The parameters define a label, an initial value, the number of characters that are visible in the field, and whether the field is read only (true means read only). If the last argument is omitted, the function creates a read-write field.

The width of the resulting element is independent of the default user interface font on the current platform.

**Example**

```dxl
DB fieldBox = create "Get Zip"
DBE zip = field(fieldBox, " Zipcode: ", ",", 12)
void unzip(DB fieldBox) {
    string zipcode = get zip
    print zipcode
} // unzip
apply(fieldBox, "Lookup", unzip)
show fieldBox
```

### richField

**Declaration**

```
DBE richField(DB box,  
    string label,  
    string initial,  
    int width  
    [,bool readOnly])
```

```
DBE richField(DB box,  
    string label,  
    richText(string initial),  
    int width)
```

```
DBE richField(DB box,  
    string label,  
    richText(string initial),  
    int width,  
    bool readOnly)
```
**Operation**

Creates a single-line rich text field element.

In the first form, arguments define a label, an initial value, the number of characters in the field, and whether the field is read only (true means read only). If the last argument is omitted, the function creates a read-write field.

The second form takes a rich text string for the initial value; it cannot create a read only field.

The third form takes a rich text string for the initial value. If readOnly is true, the function creates a read only field. If readOnly is false, the function creates a read-write field.

The width of the resulting element is independent of the default user interface font on the current platform.

---

**Declaration**

```dxl
dbe slider(DB box,  
    string label,  
    int initial,  
    int min,  
    int max)
```

**Operation**

Creates a slider element for capturing integers. The arguments passed specify a label, the initial value and the minimum and maximum values on the slider.

Sliders are best used for small ranges such as percentages. For larger numbers, or those without limits, it is better to use a text field and the `intOf` function to convert the string value to an integer.

**Example**

```dxl
DB percentBox = create "Your Feedback"
label(percentBox, "How strongly do you agree?")
DBE feelings = slider(percentBox, "Adjust slider:", 50, 0, 100)
DBE output = field(percentBox, "Output:", "", 30,  
    true)
void calc(DB percentBox) {  
    int results = get feelings  
    print results  
    set(output, results "")  
} // calc  
apply(percentBox, "Commit", calc)  
show percentBox
```
checkBox

Declaration

DBE {verticalC|c}heckBox(DB box,
    string label,
    string choices[ ],
    int initial)

Operation

Creates a set of check boxes.

Check boxes offers users choices, each of which can independently be either on or off.

The checkBox function arranges the check boxes horizontally; the verticalCheckBox function arranges them vertically. The options are passed in string array choices. The initial and returned values are bit maps indicating whether each option is checked. If the first option is checked, bit 0 is 1, if the second is checked bit 1 is 1, and so on.

Example

DB pizzaBox = create "Pizzas"
string toppings[] = {"salami", "funghi",
    "olives", "anchovies",
    "frutti di mare",
    "artichoke"}

int maxToppings = 5
DBE pizzaCheck = checkBox(pizzaBox, "Toppings:",
    toppings, 5)
bool pizzasOrdered[] = {false, false, false,
    false, false, false}

void processOrders(DB pizzaBox) {
    int bitmap = get pizzaCheck
    // bit-map of values
    int remain
    int i
    for i in 0:maxToppings do {
        remain = bitmap % 2         // remainder
        if (remain != 0) {
            pizzasOrdered[i] = true
            print toppings[i] "::"
            pizzasOrdered[i] "\n"
        } else {
            pizzasOrdered[i] = false
        }
        bitmap = bitmap / 2       // integer division
    }
apply(pizzaBox, "Order Pizzas", processOrders)
show pizzaBox

radioBox

Declaration
DBE {{verticalR|r}adioBox(DB box,
        string label,
        string choices[ ],
        int initial)

Operation
Creates a set of radio boxes.
Radio boxes offers users choices that are mutually exclusive.
The radioBox function arranges the check boxes horizontally; the verticalRadioBox function arranges them vertically. The options are passed in string array choices. The initial and returned values are indexes into that array.

Example
DB dinnerBox = create("Dinner")
string meals[] = {"Pizza", "Pasta", "Quiche",
                "Burger", "Tachos"}
DBE dinnerRadio = radioBox(dinnerBox, "Main Course: ", meals, 2)
void placeOrder(DB dinnerBox) {
    int i = get dinnerRadio
    string mealStr = meals[i]
    ack "Ordering " mealStr " now!"
} // placeOrder
apply(dinnerBox, "Order", placeOrder)
show dinnerBox

toggle

Declaration
DBE toggle(DB box,
           string label,
           bool initial)
Operation

Creates a toggle button in box with the given label and initial value.

Example

```dxl
DB parseBox = create "Simulator File Parser"
DBE binOpt = toggle(parseBox, "Use binary data", false)
show parseBox
```

date

Declaration

```dxl
DBE date(DB date_db, int width, Date init, bool calendar)
```

Operation

Creates a date/time picker control. Width specifies the width in characters of the displayed field. The variable init specifies the initial date value displayed by the control. If a null date value is supplied, the current date and time is displayed. If calendar is true, a drop-down calendar is made available in the control for selecting dates. Otherwise, up and down buttons in the control allow the user to increment and decrement values in the selected field of the control.

You can type values into the various fields of the control, and use the cursor arrow keys to select fields and increment or decrement values.

The date values are displayed according to Rational DOORS conventions: date/time values are displayed using the user’s default short date format for the current user locale, and a 24-hour clock format. Date-only values are displayed using the user’s default long date format for the current user locale.

setLimits

Declaration

```dxl
void setLimits(DBE date_dbe, Date min, Date max)
void setLimits(DBE date_dbe, AttrType type)
```

Operation

Sets the minimum and maximum limit values for a date/time picker control. If the current value displayed in the picker lies outside either of the new limits, it is updated to equal that limit. If either one of the supplied values is null, then the relevant min/max limit is not changed.

The second form sets the minimum and maximum limit values for a date/time picker control to match the limits defined for the specified attribute type. The current displayed value is updated if necessary to lie within the limit or limits.
get\text{Date}

\textbf{Declaration}\n\begin{verbatim}
Date getDate(DBE date_dbe)
\end{verbatim}\n
\textbf{Operation}\n
Returns the date value displayed in the specified DBE.

\textbf{set}

\textbf{Declaration}\n\begin{verbatim}
void set(DBE date_dbe, Date value)
void set(DBE date_dbe, string value)
\end{verbatim}\n
\textbf{Operation}\n
Updates the DBE to display the specified date value. 

The second form of the perm is updated to put the string (interpreted according to the current user locale) into the date DBE. No update occurs if the supplied string is not a valid date string.

\textbf{get}

\textbf{Declaration}\n\begin{verbatim}
string get(DBE date_dbe)
\end{verbatim}\n
\textbf{Operation}\n
Returns the displayed string in a date DBE.

\textbf{getBuffer}

\textbf{Declaration}\n\begin{verbatim}
Buffer getBuffer(DBE date_dbe)
\end{verbatim}\n
\textbf{Operation}\n
Returns the displayed string from a date DBE as a buffer.

\textbf{setFromBuffer}

\textbf{Declaration}\n\begin{verbatim}
void setFromBuffer(DBE date_dbe, Buffer b) / set(DBE,Buffer)
\end{verbatim}
Operation

Updates the DBE to display the date represented by the string in the supplied buffer, interpreted according to the current user locale. The DBE is not updated if the supplied string is not a valid date string.

Example

The following example uses the perms for the new data DBE element:

```
// DateTime Picker Test: gets and sets date values.
DB db = create "date/time picker test" // The Dialog
Date init = dateAndTime(today)          // Initial value in control
label(db,"picker:")
beside db
DBE picker = date(db,20,init,true)      // Define the control

// Callback for toggle...
void showTimeCB(DBE x)
{
    if (get(x))
    {
        set(picker,dateAndTime(getDate picker))
    }
    else
    {
        set(picker,dateOnly(getDate picker))
    }
}

// Toggle the showing of date+time or date-only
DBE showTime = toggle(db, "show time", includesTime(init))
set(showTime, showTimeCB)

// Text field to display values got from the control, and for
// sending to the control.
left db
DBE stringVal = field(db,"string field:"","",20)
```
// Get the current value from the control, as a Date value.
void getDate(DBE x)
{
    Date d = getDate(picker)
    set(stringVal,stringOf(d))
}
button(db,"Get Date",getDate)
bSIDE db

// Get the current value from the control, as a string.
void getString(DBE x)
{
    string s = get(picker)
    set(stringVal,s)
}
button(db,"Get string",getString)

// Get the current value from the control, as a Buffer.
void getDateBuffer(DBE x)
{
    Buffer b = getBuffer(picker)
    set(stringVal,b)
    delete b
}
button(db,"Get Buffer",getDateBuffer)
left db

// Update the control using a Date value
void setDate(DBE x)
{
    string s = get(stringVal)
    Date d = date(s)
    if (null d) warningBox "Not a valid date string!"
    else set(picker,d)
set(showTime, includesTime(getDate picker))

button(db, "Set Date", setDate)
beside db

// Update the control using a string value
void setString(DBE x)
{
    string s = get(stringVal)
    set(picker, s)
    set(showTime, includesTime(getDate picker))
}
button(db, "Set string", setString)

// Update the control using a Buffer value
void setDateBuffer(DBE x)
{
    Buffer b = getBuffer(stringVal)
    setFromBuffer(picker, b)
    set(showTime, includesTime(getDate picker))
    delete b
}
button(db, "Set Buffer", setDateBuffer)
left db
Date minDate = null
Date maxDate = null

// Set the minimum value accepted by the date/time picker
void setMinVal(DBE x)
{
    string s = get(stringVal)
    minDate = date(s)
    if (null minDate)
    {
    }
warningBox "Not a valid date string!"
}
else if (!null maxDate && minDate > maxDate)
{
    warningBox "Minimum date cannot be greater than maximum date."
}
else
{
    setLimits(picker,minDate,maxDate)
}
}
button(db, "Set Min from field", setMinVal)
browse db

// Set the maximum value accepted by the date/time picker
void setMaxVal(DBE x)
{
    string s = get(stringVal)
    maxDate = date(s)
    if (null maxDate)
    {
        warningBox "Not a valid date string!"
    }
else if (!null minDate && minDate > maxDate)
{
    warningBox "Maximum date cannot be less than minimum date."
}
else
{
    setLimits(picker,minDate,maxDate)
}
}
button(db, "Set Max from field", setMaxVal)
show db
Choice dialog box elements

This section defines functions and for loops that allow you to create elements that give the user a choice:

- A drop-down selector provides a simple choice.
- A combo box is an editable drop-down selector.
- A tab strip provides a simple choice where other options must be selected after the initial selection.
- Scrollable lists are a powerful mechanism for providing users with a large number of options.

These dialog box elements are all of type DBE.

choice

Declaration

DBE choice(DB box,
    string label,
    string choices[ ],
    [int noOfChoices,]
    int initial
    [,int width,
    bool canEdit])

Operation

Creates a drop-down selector. This shows only the current value until the user clicks in it, when the whole range is displayed. The initial argument specifies which value is selected by default, counting from 0.

The string array choices must have been declared at a fixed size, with each element containing a string. The optional noOfChoices argument specifies the number of elements of the choices array that contain real choices.

The optional width argument specifies the number of characters in the choice box. When used, this argument must be followed by a boolean value to indicate whether the choice can be edited by the user. If canEdit is true, the choice box is editable (a combo box). If width is 0, -1, or omitted, the standard width is used.

The width of the resulting element is independent of the default user interface font on the current platform. The width will be consistent with the legacy behavior on Western platforms with regard to the resultant width calculated from the specified number of characters.

Example

DB reqBox = create "Edit Requirement"

string importance[] = {"Vital", "Useful",
                        "Convenient", "Useless"}

DBE reqImport = choice(reqBox, "Importance: ",
                         importance, 2)
void accept(DB reqBox) {
    int i = get reqImport
    print importance[i]
} // accept
ok(reqBox, "Accept", accept)
show reqBox

void accept(DB reqBox) {
    int i = get reqImport
    print importance[i]
} // accept
ok(reqBox, "Accept", accept)
show reqBox

Declaration

DBE tab(DB box, string choices[ ] [,int noOfChoices]
[ [,int width,
    int height],
    void (DBE theTab)])

Operation

Creates a tab strip. This function behaves much like the list function.

The string array choices must have been declared at a fixed size, with each element containing a string. The optional noOfChoices argument specifies the number of elements of the choices array that contain real choices.

The optional width and height arguments specify the initial size of the tab strip in pixels. If width and height are not specified, the size is controlled by the elements the tab strip contains, or from the form if the tab strip is connected to it. If the right edge of a tab strip is to remain unattached, you must specify a size. A tab strip with an initial size can stretch if placement constraints are incompatible with the size specified.

You can place other dialog box elements inside a tab strip using the placement keyword inside, but you should not put an element with no innate size (like a list box) inside a tab with no innate size. For further information on tab strip placement, see “Attachment placement,” on page 601.

The callback function must identify which tab has been selected.

Example

DB box = create "Test"
void tabSelected(DBE theTab){
    int i = get theTab
}
string items[] = {"A", "B", "C"}
DBE theTab = tab(box, items, 300, 400, tabSelected)
theTab->"top"->"form"
theTab->"left"->"form"
theTab->"bottom"->"form"
list

**Declaration**

DBE list(DB box,  
    string label,  
    [int width,]  
    int visible,  
    string values[ ]  
    [,int noOfValues])

**Operation**

Creates a list element containing the given values, from which the user can choose at most one item. If there are many or a variable number of options, a list is better than a choice as it does not attempt to display more than the number of items passed in the `visible` argument. If the `width` argument is present, the element is created at the specified size in pixels. Otherwise, the list is created to use the full width of the dialog box.

You can supply either a complete array of strings, such as a constant array, or a partially filled array, with the number of items supplied in the `noOfValues` argument. You can create a list with initially no entries by setting `noOfValues` to 0, although you must still supply a valid string array.

Note that there is no initial selection; to do this, use the `set(value or selection)` function. You can also define callbacks for lists.

**Example**

DB coffeeBox = create "Coffees"

string coffees[] = {"Mocha", "Sumatra Blue",  
    "Jamaica Mountain",  
    "Mysore", "Kenya", "Java"}

DBE coffeeList = list(coffeeBox, "Choose one of:", 5, coffees)

void getCoffees(DBE coffeeList) {
    int i = get coffeeList
    if (i == 0) ack "Mmm, Mocha..."
    if (i == 5) ack "Watch out for trademark violations"
} // getCoffees

// run callback directly upon list selection
set(coffeeList, getCoffees)
show coffeeBox
multiList

Declaration

DBE multiList(DB box,
            string label,
            [int width,]
            int visible,
            string values[ ]
            [,int noOfValues])

Operation

Creates a list element containing the given values, from which the user can choose one or more items. In all other respects
this function is exactly the same as the list function.

Example

DB attrShow = create "Attributes"
string attrNames[100]
int noOfAttrs = 0
string an
if (null current Module) {
    ack "Please run this function from a module"
    halt
}
for an in current Module do
    attrNames[noOfAttrs++] = an
DBE attrList = multiList(attrShow, "Attributes:",
                        5, attrNames, noOfAttrs)
void printAttrs(DB box) {
    string attrName
    for attrName in attrList do {
        print attrName " = " ((current Object).attrName) "\n"
    }
} // printAttrs
apply(attrShow, "Print", printAttrs)
void clearSelection(DB box) {
    int i
    for i in 0:noOfAttrs do
        set(attrList, i, false)
} // clearSelection
apply(attrShow, "Clear", clearSelection)
show attrShow

selectedElems

**Declaration**

```c
int selectedElems(DBE listView)
```

**Operation**

Returns the number of elements currently selected in the specified list view.

Typically this is either 0, 1 or a positive integer (if the list view was created using the `listViewOptionMultiselect` style).

If the DBE is not a list view, a run-time error occurs.

---

**for value in list (selected items)**

**Syntax**

```c
for s in list do {
  ...
}
```

where:

- `s` is a string variable
- `list` is a multilist of type DBE

**Operation**

Assigns the string `s` to be each successive selected item in a multilist, `list`.

**Example**

```c
string at
for at in attrList do print at " is selected\n"
```

---

**for position in list (selected items)**

**Syntax**

```c
for i in list do {
  ...
}
```
where:

\( i \) is an integer variable

\( \text{list} \) is a multilist of type DBE

**Operation**

Assigns the integer \( i \) to be the index of each successive selected item in a multilist, \( \text{list} \).

**Example**

```dxl
int totalWeight = 0
int index
for index in components do
    totalWeight += compWeights[index]
```

---

**View elements**

This section defines functions and for loops that allow you to create list views and tree views in your dialog boxes.

**Drag-and-drop**

Drag-and-drop operations are possible in list views and tree views, provided a callback function is specified when the list view or tree view is created. The callback takes the form:

```dxl
void callback(DropEvent dropEvent)
```

The DropEvent structure is unique to the source of the drag; it exists for only as long as the dialog box element being dragged.

Properties are defined for use with the . (dot) operator and DropEvent structure to extract information about drop events, as shown in the following syntax:

```dxl
dropEvent.property
```

where:

- `dropEvent` is a variable of type DropEvent
- `property` is one of the drag-and-drop properties

The following tables list the properties and the information they extract:

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourcePath</td>
<td>The path of the source item of a drag operation; this is only valid if <code>sourceIsListView</code> is true, otherwise, it is null.</td>
</tr>
</tbody>
</table>
### String property

| targetPath | The path of the target item of a drag operation; this is only valid if targetIsListView is true, otherwise, it is null. |

### Boolean property

| sourceIsTreeView | Whether the source of the drag is a tree view. |
| sourceIsListView | Whether the source of the drag is a list view. |
| targetIsTreeView | Whether the target of the drag is a tree view. |
| targetIsListView | Whether the target of the drag is a list view. |

### Integer property

| sourceIndex | The index of the source item of a drag operation; this is only valid if sourceIsListView is true, otherwise, it is -1. |
| targetIndex | The index of the target item of a drag operation; this is only valid if targetIsListView is true, otherwise, it is -1. |

### DBE property

| source | The source dialog box element of the drag operation; this is always the element for which the callback was defined. |
| target | The target dialog box element of the drag operation. |

### Example

```dxl
DropEvent de
bool b = de.targetIsTreeView
DBE testList = de.source
```
listView

Declaration

DBE listView(DB box
    [,void callback(DropEvent event],
    int options,
    int width,
    int lines,
    string items[ ]
    [,int noOfItems])

Operation

Creates a list view having the specified width in pixels and with the specified number of lines.

The optional callback function enables the list view to participate in drag-and-drop events. When this list view is the source of a drop operation, the callback fires and the DropEvent structure can be queried. For further information, see “Drag-and-drop,” on page 545. If the callback function is not supplied, the user cannot use drag-and-drop in the list view.

The string array items must have been declared at a fixed size, with each element containing a string. The optional noOfItems argument specifies the number of elements of the items array that contain real choices.

The argument options controls whether the list view has check boxes. The value can be one of the following:

- listViewOptionCheckboxes provides check boxes
- listViewOptionMultiselect makes it possible to select more than one item*
- 0 no check boxes or multi-select capability
- listViewOptionCheckboxes|listViewOptionMultiselect provides check boxes and multi-select capability
- listViewOptionSortText for use with the setSortColumn perm

*Beginning in Rational DOORS version 9.6.1.7, you can use Ctrl+A keys to select all items in a multiselect listView.

deleteColumn

Declaration

string deleteColumn(DBE listView,
    int columnIndex)

Operation

Deletes from listView the column identified by columnIndex counting from 0. This works only for list views.
insertColumn(list view)

Declaration

void insertColumn(DBE listView,
    int columnIndex,
    string title,
    int width,
    Icon icon)

Operation

Inserts a column in list view after the column identified by columnIndex counting from 0. The new column has title title, width in pixels of width, and icon icon. To insert a column without an icon use the value iconNone. For other valid icon values, see “Icons,” on page 483.

This works only for list views.

c getColumnValue

Declaration

string getColumnValue(DBE listView,
    int row,
    int column)

Operation

Returns the value of the item or subitem identified by row in column. Both row and column count from 0. This works only for list views.

Example

This example returns the 34th row of the first column in list view main.

string s = getColumnValue(main, 33, 0)

getCheck

Declaration

bool getCheck(DBE listView,
    int index)

Operation

Returns true if the check box identified by index is selected; otherwise, returns false. This works only for list views.
setCheck

Declaration

void setCheck(DBE listView,  
    int index,  
    bool checked)

Operation

Selects or clears the check box identified by index according to the value of checked. This works only for list views.

getSortColumn

Declaration

int getSortColumn(DBE listView)

Operation

Returns the column in listView that is being sorted. This works only for list views.

setSortColumn

Declaration

void setSortColumn(DBE listView,  
    int columnIndex)

Operation

Sets the column to be sorted to the column specified by columnIndex. This works only for list views.

treeView

Declaration

DBE treeView(DB box  
    [,void callback(DropEvent event)],  
    int options,  
    int width,  
    int visible)

Operation

Creates a tree view having the specified width in pixels and with the specified number of visible items (which controls the height of the tree view).
The optional callback function enables the list view to participate in drag-and-drop events. When this list view is the source of a drop operation, the callback fires and the DropEvent structure can be queried. For further information, see “Drag-and-drop,” on page 545. If the callback function is not supplied, the user cannot use drag-and-drop in the list view.

The options argument can be 0 or treeViewOptionSorted, which sorts the tree view.

exists(tree view)

Declaration

bool exists(DBE treeView,
            string fullPath)

Operation

Returns true if a fullPath is the full path name of a tree view; otherwise, returns false.

layoutDXL

Declaration

void layoutDXL(DBE treeView)

Operation

 Loads the specified tree view with a hierarchy of DXL files, which can be used for column layout DXL.
 If the DBE is not a tree view, a run-time error occurs.

attributeDXL

Declaration

void attributeDXL(DBE treeView)

Operation

 Loads the specified tree view with a hierarchy of DXL files, which can be used for DXL attribute.
 If the DBE is not a tree view, a run-time error occurs.

getDXLFileHelp, getDXLFileName

Declaration

string getDXLFileHelp(DBE treeView)
string getDXLFileName(DBE treeView)
Operation
These functions assume that the specified tree view contains a hierarchy of DXL files loaded using the layoutDXL function (similar to the contents of the DXL Browser dialog box). If one of the files is selected, and you call either of these functions, typically from a button callback, they behave as described here.

The first function returns the help text associated with the selected DXL file.

The second function returns the name of the selected file.

If the dialog box element is not a tree view, a run-time error occurs.

templates

Declaration

void templates(DBE treeView)

Operation

Populates the specified tree view with a hierarchy of available templates (DXL files) that are in the lib\dxl\standard\ template directory.

getTemplateFileName

Declaration

string getTemplateFileName(DBE treeView)

Operation

Assumes that the specified tree view contains a hierarchy of available templates previously loaded using the templates function.

When a template is selected on a user’s PC or workstation, returns the full path of the selected file. Otherwise, returns a null string.

Example

// prevent dxl timeout dialog
pragma runLim, 0
// constants
const int INITIAL_TREE_WIDTH  = 500
const int INITIAL_TREE_HEIGHT = 20
// dxl dialog
DB dlg = null
// dxl elements
DBE dbeTree, dbeLabel
// function
void fnTreeSelect(DBE xx)
{
    string sTemplate = getTemplateFileName(xx)
    // only relevant if actual template was
    // selected
    if (sTemplate != null)
    {
        // inform user
        infoBox(dlg, sTemplate)
    }
}

dlg = create(dbExplorer, "Templates", styleCentered | styleFixed)
    // label
dbeLabel = label(dlg, "Please select an item from
    the tree...")

dbeLabel->"top"->"form"
dbeLabel->"left"->"form"
    dbeLabel->"right"->"unattached"
    dbeLabel->"bottom"->"unattached"
// tree view
dbeTree = treeView(dlg, 0, INITIAL_TREE_WIDTH, INITIAL_TREE_HEIGHT)
    dbeTree->"top"->"spaced"->dbeLabel
    dbeTree->"left"->"form"
    dbeTree->"right"->"form"
    dbeTree->"bottom"->"form"
realize dlg
{
    // callbacks
    set(dbeTree, fnTreeSelect)
    // load templates into tree view
    templates(dbeTree)
}
block dlg
for value in list view (selected items)

Syntax

for s in listView do {
    ...
}

where:

s is a string variable

listView is a list view of type DBE

Operation

Assigns the string s to be each successive selected item in a list view.

for position in list view (selected items)

Syntax

for i in listView do {
    ...
}

where:

i is an integer variable

listView is a list view of type DBE

Operation

Assigns the integer i to be the index of each successive selected item in a list view, listView.

Text editor elements

This section defines text editor functions, which allow you to create a full function text editing panel in your dialog box. These have the same functions as all Rational DOORS text panels, including pop-up menu support for loading and saving files.
text(box)

Declaration

DBE text(DB box,
    string label,
    string initial,
    [int width,]
    int height,
    bool readOnly)

Operation

Creates a multi-line text element in the dialog box box. The arguments define a label, an initial value, the width of the text box in pixels, the height of the text box in pixels, and whether the text box is read only (true means the user cannot modify the contents of the text box). If width is omitted, the box takes the full width of the window.

Example

void sendRID(DB RIDbox) {
    // process RID in some way
} // sendRID

DB RIDbox = create "Review Item Discrepancy"

DBE response = text(RIDbox, "Your response:", ",",
    200, false)

apply(RIDbox, sendRID)

show RIDbox

richText(box)

Declaration

DBE richText(DB box,
    string label,
    {string
       initial|richText(string initial)},
    int width,
    [int height,]
    bool readOnly)

Operation

Creates a multi-line rich text element in the dialog box box. The arguments define a label, an initial value (which can be rich text), the width of the text box in pixels, the height of the text box in pixels, and whether the text box is read only (true means the user cannot modify the contents of the text box). If height is omitted, the box takes the full height of the window.

If the blinking cursor appears at the end of the text in the box when it is displayed, append “” to the end of the rich text string before passing it to the perm.
home

Declaration
void home(DBE textElem)

Operation
Causes the cursor to go to the first character in textElem.

Example
home messageArea

modified

Declaration
bool modified(DBE textElem)

Operation
Returns true if the text in textElem has been modified since it was last set.

Example
if (modified errorLog && confirm "Save error log changes?")
   saveErrorLog

get(selected text)

Declaration
bool get(DBE textElem,
        int &first,
        int &last)

Operation
Returns the selection indices for a text element. If there is a selection, the function returns true, and sets the first and last arguments to the zero-based indices of the first character and the character immediately after the last one selected.

If there is no selection, the function returns false.

Example
DB splitBox = create "Text splitter"
DBE objTextElem = text(splitBox, "Object text:",
                        "1234567890", 200, false)
void getSelection(DB splitBox) {
    int first, last
    if (get(objTextElem, first, last)) {
        string ot = get objTextElem
        string selection = ot[first:last-1]
        print "You selected:\n" selection "\n"
    } else {
        print "No selection\n"
    }
} // getSelection

apply(splitBox, "Get selection", getSelection)

show splitBox

Buttons

This section defines functions that allow you to create buttons on dialog boxes. Rational DOORS dialog boxes provide two kinds of buttons: those across the bottom of the dialog box, and those that appear in the dialog box area itself.

ok

Declaration

DBE ok(DB box, 
    [string label,]
    void callback(DB))

Operation

Adds a button to the row of standard buttons on the dialog box, and associates it with the given callback function. If the label argument is passed, the button has that label; otherwise it has the standard label OK.

When the user clicks the button, the function is called with the parent dialog box as the argument, and the dialog box is removed from the screen.

Example

void writeout(DB box) {
    // user code here
} // writeout

ok(fileOpBox, "Write", writeout)
apply

Declaration

DBE apply(DB box,
    [string label,]
    void callback(DB))

Operation

Adds a button to the row of standard buttons on the dialog box, and associates it with the given callback function. If the label argument is passed, the button has that label; otherwise it has the standard label Apply.

When the user clicks the button, the function is called with the parent dialog box as the argument. The dialog box remains on the screen, enabling this or other buttons to be clicked.

Example

void sumAttrs(DB box) {
    // user code here
}

apply(analysisBox, "Calculate", sumAttrs)

close

Declaration

void close(DB box,
    bool includeButton,
    void closeAction(DB))

Operation

Normally a Close button is added to the row of standard buttons on a dialog box. The normal action of the Close button is to close the dialog box.

If the includeButton argument is false, the Close button is omitted from the dialog box, although the user can still close the window via the window manager or system menu. This enables you to supply a close-action button that has an alternative label.

Because closing the dialog box might not always be desirable behavior, this function enables you to replace the standard close action with a callback function. When a callback function is supplied, windows are not automatically closed; the callback must explicitly hide the dialog box (see the hide(dialog box) function).

Example

DB exBox = create "Example"
DBE tp = text(exBox, "Text", "Type in here", 100,
    false)
DBE check = toggle(exBox, "Check before closing",
    true)
void checkText(DB exBox) {
    if (modified(tp) && !confirm("Text modified, really close?"))
        return
    hide exBox
} // checkText

ok(exBox, "Cancel", checkText)
close(exBox, false, checkText)
show exBox

button

Declaration

DBE button(DB box, string label, void callback(DBE))
    [, bool variableSize | int style])

DBE button(DB box, string label, void callback [, bool variableSize | int style [, int width]])

Operation

Creates a button in the specified dialog box. The callback function fires whenever the user clicks on the button.

The button can have either a label or an arrow symbol defined by one of the following constants in “ok, apply, button(arrows),” on page 559.

The optional fourth argument enables you to specify the size or style of the button.

If variableSize is false, the button is 50 pixels wide by 13 pixels high. If variableSize is missing or true, the button size depends on the label.

The possible values for style are: styleIsDefault, styleIsCloseBtn, styleStandardSize, or any OR combination of these values.

The second variant has an optional width argument that enables the user to specify the width of the button, in pixels. (As a guide, standardSize buttons are 50 pixels wide.) This argument has no effect if the variableSize argument is specified as false, or if the style argument is specified and includes the standardSize option.

Example

DB resultsBox = create "Summary Results Display"

DBE caption

void repaint(DBE canv) {
    background(canv, colorLightBlue)
    color(canv, colorMaroon)
    string cap = get caption
    draw(canv, 100, 50, cap)
} // repaint
DBE canv = canvas(resultsBox, 400, 100, repaint)
caption = field(resultsBox, "Caption:",
        "callbacks will plot data", 20)
browse resultsBox
void trendPlot(DBE calledfrom) {
    // user code here
    repaint canv
} // trendPlot
DBE trends = button(resultsBox, "Show trends", trendPlot)
show resultsBox

ok, apply, button(arrows)

Both standard and dialog-area buttons can be created with an arrow instead of a text label. To do this, replace the string
label with one of the following constants:
topleftArrow
topArrow
topRightArrow
topLeftArrow
topRightArrow
leftArrow
leftRightArrow
bottomLeftArrow
downArrow
bottomRightArrow
rightArrow
upArrow
upDownArrow

Example
DB arrowBox = create "Arrow Demo"
void doNothing(DBE x) {}  
void doNothing(DB x) {}
DBE tl = button(arrowBox, topleftArrow, doNothing)
browse arrowBox
DBE up = button(arrowBox, upArrow, doNothing)
DBE tr = button(arrowBox, topRightArrow, doNothing)
Canvases

This section defines functions for canvases, which allow DXL programs to draw graphics, such as charts and diagrams, in dialog boxes.

Any graphics layout DXL should always specify all co-ordinates in drawing units (du), for example:

```
rectangle(myCanv, 10 du, 10 du, 20 du, 20 du)
```

Otherwise the graphics do not print properly.

Keyboard event constants

**Declaration**

```
const char keyInsert
const char keyDelete
const char keyHome
const char keyEnd
const char keyPageUp
const char keyPageDown
```
const char keyUp
const char keyDown
const char keyLeft
const char keyRight
const char keyHelp
const char keyF1
const char keyF2
const char keyF3
const char keyF4
const char keyF5
const char keyF6
const char keyF7
const char keyF8
const char keyF9
const char keyF10
const char keyF11
const char keyF12

Operation
These are character constants that represent keyboard presses for invisible characters. They are returned by callbacks defined using set.

canvas

Declaration
DBE canvas(DB box,
    int width,
    int height,
    void repaint(DBE art))

Operation
Creates a drawing surface, which can be used for graphical output with the DXL Graphics Library functions.

Graphics must only be directed to the canvas from the callback function, repaint, which you must define, otherwise they are lost at the next repainting. The function is called back when the window appears on the screen, when it is de-iconified, or when an overlapping window is moved.

To add a mouse or key callback to a canvas, use the set(select) function.

Example
DB artBox = create "Try resizing this window"
void doDrawing(DBE art){
   // repaint callback function
   int i, x, y, w, h
   int cw = width art
   int ch = height art

   for i in 0 : 15 do {
      color(art, i)
      x = random cw // size graphics to canvas
      y = random ch
      w = (cw - x) / 2
      h = (ch - y) / 2
      rectangle(art, x, y, w, h)
   } }

DBE art = canvas(artBox, 400, 300, doDrawing)
show artBox

background

Declaration

void background(DBE canvas,
               int colorNo)

Operation

Colors the whole of the canvas with the given color. For information on valid color numbers, see “Logical colors,” on page 591. This function destroys any existing drawing, and is equivalent to drawing a rectangle the size of the canvas. This function is recommended if you wish to color the whole canvas or erase the current image.

Example

DB graphBox = create "Graphics"
void repaint(DBE graph) {
   background(graph,
             logicalMediumIndicatorColor)
   // draw picture here
} // repaint

DBE graph = canvas(graphBox, 250, 75, repaint)
show graphBox

realBackground

Declaration

void realBackground(DBE canvas,
                 int colorNo)
Operation
Colors the whole of the canvas with the given color. For information on valid color numbers, see “Real colors,” on page 593. This function destroys any existing drawing, and is equivalent to drawing a rectangle the size of the canvas. This function is recommended if you wish to color the whole canvas or erase the current image.

Example
DB colorBox = create "To demonstrate the colors"

void doDrawing(DBE colorCanvas) {
    // repaint callback function
    // background(art, logicalPageBackgroundColor)
    realBackground(colorCanvas, realColor_Black)
    color(colorCanvas, logicalPageBackgroundColor)
    draw(colorCanvas, 15, 15, "logicalPageBackgroundColor")
    realColor(colorCanvas, realColor_Green)
    draw(colorCanvas, 15, 45, "Red")
    realColor(colorCanvas, realColor_Magenta)
    draw(colorCanvas, 15, 60, "Magenta")
}

DBE colorCanvas = canvas(colorBox, 400, 300, doDrawing)
show colorBox    // draw picture here

color

Declaration
void color(DBE canvas, int colorNo)

Operation
Sets the drawing color for the canvas to be the given colorNo. For information on valid color numbers, see “Logical colors,” on page 591.

Example
color(board, logicalDataTextColor)

realColor

Declaration
void realColor(DBE canvas, int realColor)
**Operation**

Sets the drawing color for the canvas to be the given `realColor`. For information on valid color numbers, see “Real colors,” on page 593.

**Example**

See the example for the `realBackground` function.

---

### font

**Declaration**

```dxl
void font(DBE canvas,
          int level,
          int mode)
```

**Operation**

Sets the font for drawing strings on the canvas. The font is specified by two logical values corresponding to those in the Font Options window. The `level` argument is in the range 1 to 9 to represent the level in the tree at which a node appears. Essentially, this argument controls the size; level 1 is the top level of heading and typically appears in a large typeface. The `mode` argument controls which font is used: 0 sets body font, 1 sets heading font, and 2 sets graphics font. You can also use the constants `HeadingsFont`, `TextFont`, and `GraphicsFont`.

**Note:** The actual font size and typeface depend on the user’s settings.

**Example**

```dxl
DB graphBox = create "Graphics"
void repaint(DBE graph) {
    background(graph, logicalPageBackgroundColor)
    color(graph, logicalDataTextColor)
    int x = 10
    int fsize
    for fsize in 1:9 do {
        font(graph, fsize, 0)
        draw(graph, x, 20, fsize "")
        font(graph, fsize, 1)
        draw(graph, x, 60, fsize "")
        font(graph, fsize, 2)
        draw(graph, x, 90, fsize "")
        x += 20
    }
} // repaint
DBE graph = canvas(graphBox, 300, 100, repaint)
show graphBox
```
height

Declaration

int height(DBE canvas
           [,string s])

Operation

With a single argument, this returns the height of canvas. This function must be used in repaint functions to obtain the size of the area into which to draw, as this might change.

When the second argument is passed, the function returns the height of the space required to render the string s in the current font.

Example

This example obtains the height of the canvas:

int h = height board

width

Declaration

int width(DBE canvas
           [,string s])

Operation

With a single argument, this returns the width of a canvas element. This function must be used in repaint functions to obtain the size of the area into which to draw, as this might change.

When the second argument is passed, the function returns the width of the space required to render the string s in the current font.

Example

This example obtains the width of the canvas:

int w = width board

This example obtains the height and width of the string variable message:

DB graphBox = create "Graphics"

void repaint(DBE graph) {
            background(graph,
                        logicalMediumIndicatorColor)
            color(graph, logicalHighIndicatorColor)
            int w = width graph
            int h = height graph
            string message = w " by " h ""
            int tw = width(graph, message)
int th = height(graph, message)
int x = (w - tw)/2
int y = (h - th)/2
draw(graph, x, y, message)
} // repaint

DBE graph = canvas(graphBox, 250, 150, repaint)
show graphBox

---

rectangle

Declaration

void rectangle(DBE canvas,
    int x,
    int y,
    int w,
    int h)

Operation

Draws a rectangle filled with the current color at position (x,y), width w, height h on canvas. The co-ordinate system has its origin at the top left.

Example

DB graphBox = create "Graphics"
void repaint(DBE graph) {
    background(graph, logicalMediumIndicatorColor)
    color(graph, logicalHighIndicatorColor)
    rectangle(graph, 50, 50, 150, 50)
} // repaint

DBE graph = canvas(graphBox, 250, 150, repaint)
show graphBox

---

box

Declaration

void box(DBE canvas,
    int x,
    int y,
    int w,
    int h)
Operation

Draws an outline rectangle with the current color at position \((x, y)\), width \(w\), height \(h\) on canvas. The co-ordinate system has its origin at the top left.

Example

DB graphBox = create "Graphics"
void repaint(DBE graph) {
    background(graph,
        logicalMediumIndicatorColor)
    color(graph, logicalHighIndicatorColor)
    box(graph, 50, 50, 150, 50)
} // repaint
DBE graph = canvas(graphBox, 250, 150, repaint)
show graphBox

line

Declaration

void line(DBE canvas,
    int x1,  
    int y1,  
    int x2,  
    int y2)

Operation

Draws a line from \((x_1, y_1)\) to \((x_2, y_2)\) in the current color. The co-ordinate system has its origin is at top left.

Example

DB graphBox = create "Graphics"
void repaint(DBE graph) {
    background(graph,
        logicalMediumIndicatorColor)
    color(graph, logicalHighIndicatorColor)
    line(graph, 0, 0, width graph, height graph)
} // repaint
DBE graph = canvas(graphBox, 250, 150, repaint)
show graphBox
**ellipse**

**Declaration**

```dxl
void ellipse(DBE canvas,
    int x,
    int y,
    int w,
    int h)
```

**Operation**

Draws an ellipse filled with the current color within the bounding box specified by position \((x, y)\), width \(w\), height \(h\) on \(canvas\). The co-ordinate system has its origin at the top left. If \(w\) and \(h\) are the same, this draws a circle.

**Example**

```dxl
DB graphBox = create "Graphics"
void repaint(DBE graph) {
    background(graph, logicalMediumIndicatorColor)
    color(graph, logicalHighIndicatorColor)
    ellipse(graph, 0, 0, width graph, height graph)
} // repaint
DBE graph = canvas(graphBox, 250, 150, repaint)
show graphBox
```

**draw**

**Declaration**

```dxl
void draw(DBE canvas,
    int x,
    int y,
    string s)
```

**Operation**

Draws the string \(s\) at position \((x, y)\), in the current color with the current font. The co-ordinate system has its origin at top left. The vertical position of the text is at the baseline of the font, so the co-ordinates must be the position for the bottom of most characters. Characters with a descender, such as \(g\), use height above and below the baseline.

**Example**

```dxl
DB graphBox = create "Graphics"
void repaint(DBE graph) {
```
background(graph, logicalMediumIndicatorColor)
color(graph, logicalHighIndicatorColor)
string message = (width graph) " by " (height graph) ""
draw(graph, 10, 20, message)
} // repaint
DBE graph = canvas(graphBox, 250, 150, repaint)
show graphBox

drawAngle

**Declaration**
void drawAngle(DBE canvas, int x, int y, string s, real angle)

**Operation**
Draws the string s rotated counter-clockwise by the given angle (in degrees). The rotation is centered around the baseline of the font, at the start of the string.

**Example**
string message = "Hello world"
real angle
DB graphBox = create "drawAngle test"
void repaint (DBE graph) {
  background(graph, logicalMediumIndicatorColor)
  color(graph, logicalHighIndicatorColor)
  font(graph, 1, 1)
  draw(graph, 0, 25, message)
  for (angle = 0.0; angle < 360.0; angle += 360.0 / 8.0)
    drawAngle(graph, 130, 125, message, angle)
}
DBE graph = canvas(graphBox, 300, 250, repaint)
show graphBox
polarLine

Declaration

void polarLine(DBE myCanvas,
    int x,
    int y,
    int lineLength,
    int lineAngle)

Operation

Draws a line on the specified canvas from the co-ordinates (x, y), with a length of lineLength at an angle of lineAngle degrees to the horizontal. The horizontal starts at the 3 o'clock position, and the angle increases in a clockwise direction.

Example

int offset = 0
void doDrawing(DBE myCanvas) {
    int i = 0
    ellipse(myCanvas, 50, 50, 200, 200)
    for (i = 0; i < 360; i += 20) {
        polarLine(myCanvas, 150, 150, 100, i +
                    offset)
    }
    offset++
    if (offset >= 20) offset = 0
}

DB myWindow = create "Example"
DBE myCanvas = canvas(myWindow, 300, 300, doDrawing)
show myWindow

polygon

Declaration

void polygon(DBE myCanvas,
    int coordArray[ ])

Operation

Draws a polygon on the specified canvas using successive co-ordinates held in the specified array.
Example

```dxl
void doDrawing(DBE myCanvas) {
    int count = 6
    int coords[8]

    background(myCanvas,
               logicalPageBackgroundColor)

    color(myCanvas, logicalDataTextColor)
    coords[0] = 20
    coords[1] = 20
    coords[2] = 100
    coords[3] = 30
    coords[4] = 200
    coords[5] = 100
    coords[6] = 80
    coords[7] = 150
    polygon(myCanvas, coords)
}
```

```dxl
DB myWindow = create "Example"

DBE myCanvas = canvas(myWindow, 300, 300,
                       doDrawing)
```

```dxl
show myWindow
```

bitmap

Declaration

```dxl
void bitmap(DBE myCanvas,
            string fileName,
            int x, int y)
```

Operation

Draws the bitmap stored in the specified file, at co-ordinates \((x,y)\) on the specified canvas. This is functionally equivalent to calling loadBitmap, drawBitmap and destroyBitmap.

loadBitmap

Declaration

```dxl
Bitmap loadBitmap(DBE myCanvas,
                  string fileName,
                  bool colorMap,
                  int& w, int& h)
```
Operation

Loads and caches, for the canvas `myCanvas`, the bitmap stored in file `fileName`.

If `colorMap` is `true` a private color map is used; otherwise, the system color map is used.

Returns in `w` and `h` the width and height of the bitmap.

Returns the handle of the bitmap.

drawBitmap

Declaration

```c
void drawBitmap(DBE myCanvas,
    Bitmap myBitMap,
    int x,
    int y)
```

Operation

Draws the specified bitmap on the specified canvas at co-ordinates `(x,y)`.

destroyBitmap

Declaration

```c
void destroyBitmap(DBE myCanvas,
    Bitmap bitMapHandle)
```

Operation

Destroys the specified bitmap cached for the canvas `myCanvas`.

export

Declaration

```c
void export(DBE myCanvas,
    string fileName,
    string formatName)
```

Operation

Exports the specified canvas to the specified file in the specified format, which can be one of these values:

<table>
<thead>
<tr>
<th>formatName</th>
<th>Format</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;EPS&quot;</td>
<td>Encapsulated PostScript®</td>
<td>All</td>
</tr>
<tr>
<td>&quot;EMF&quot;</td>
<td>Enhanced Metafile</td>
<td>Windows</td>
</tr>
</tbody>
</table>
**print**

**Declaration**

```dxl
void print(DBE myCanvas,
           real hScale,
           real vScale)
```

**Operation**

Prints the specified canvas, horizontally scaled by $hScale$ and vertically scaled by $vScale$. The width of the printed image is $hScale$ times the width of the on-screen image. The height of the printed image is $vScale$ times the height of the on-screen image.

**startPrintJob, endPrintJob**

**Declaration**

```dxl
void startPrintJob(string title)
void endPrintJob()
```

**Operation**

This enables you to package up several prints into one job, to avoid having the Print dialog box shown repeatedly.

**Example**

```dxl
// Canvas printing demo
int counter = 1
DB theBox = centered "Canvas print demo"
DBE tog, canv
void repaint(DBE canv) {
  realBackground(canv, realColor_White)
  font(canv, 1,1)
  draw(canv, 150, 150, "This is page " counter "")
}
```
void getSettings() {
    bool b = get tog 
    showPrintDialogs b
}

void printOne(DB xx) {
    getSettings
    counter = 1
    print(canv, 1.0, 1.0)
}

void printThree(DB xx) {
    getSettings
    startPrintJob "Batch print job"
    counter = 1
    print(canv, 1.0, 1.0)
    counter = 2
    print(canv, 1.0, 1.0)
    counter = 3
    print(canv, 1.0, 1.0)
    endPrintJob
    counter = 1
}

canv = canvas(theBox, 400, 400, repaint)
tog = toggle(theBox, "Show dialogues", showPrintDialogs())
apply(theBox, "Print one", printOne)
apply(theBox, "Print three", printThree)
show theBox

Complex canvases

This section defines functions for dialog box canvases, which support all the functions of the standard Rational DOORS windows, such as in-place editing, tool tips, header bars, scroll bars, menu bars, status bars, tool bars, and tool bar combo boxes. Normally, canvases do not have these dialog box elements, but functions are available to implement them.

In-place editing

There are three types of in-place editors available in each canvas. They are selected through the following constants:

inPlaceString specifies a line editor
inPlaceText specifies a text editor
In-place editing is normally disabled on canvases. The `hasInPlace` function defines an in-place edit callback function, and enables the editors.

---

### hasInPlace

**Declaration**

```c
void hasInPlace(DBE da, void cb(DBE el, event))
```

**Operation**

The callback function is called on one of two event types. The event type is the second argument passed to callback function with one of the following values:

- `inPlaceTextFilled` the text box is full and needs to be expanded
- `inPlaceTextChange` the text box contents has been modified

---

### inPlaceMove

**Declaration**

```c
void inPlaceMove(DBE da, editor, int x, int y, int w, int h)
```

where:

- `editor` is one of the in-place editors: `inPlaceString`, `inPlaceText`, or `inPlaceChoice`

**Operation**

Moves the specified type of editor to the given location within the canvas.

---

### inPlaceShow

**Declaration**

```c
void inPlaceShow(DBE da, editor, bool showing)
```

where:

- `editor` is one of the in-place editors: `inPlaceString`, `inPlaceText`, or `inPlaceChoice`
Operation
Displays or hides the specified type of editor, at the location defined by the inPlaceMove function, depending on the value of showing.
This function automatically triggers the repaint callback.

inPlaceChoiceAdd

Declaration
void inPlaceChoiceAdd(DBE da, string item)

Operation
Adds an option to the in-place choice editor.

inPlaceCut, inPlaceCopy, inPlacePaste

Declaration
void inPlace{Cut|Copy|Paste}(DBE da, editor)
where:

editor is one of the in-place editors: inPlaceString, inPlaceText, or inPlaceChoice

Operation
Perform cut, copy, or paste operations on the contents of the in-place text or string editor.

inPlaceGet

Declaration
{string|int} inPlaceGet(DBE da, editor)
where:

editor is one of the in-place editors: inPlaceString, inPlaceText, or inPlaceChoice

Operation
Returns the in-place editor specified by editor. The return value is a string for the text or string editors, and an integer for the choice editor.
inPlaceSet

Declaration

void inPlaceSet(DBE da, editor, {string s|int i})

where:

editor is one of the in-place editors: inPlaceString, inPlaceText, or inPlaceChoice

Operation

Sets the text or string editor to have the value s, or sets the choice editor to have the value i.

inPlaceReset

Declaration

void inPlaceReset(DBE da, editor)

where:

editor is one of the in-place editors: inPlaceString, inPlaceText, or inPlaceChoice

Operation

Resets the specified editor to have no value.

inPlaceTextHeight

Declaration

int inPlaceTextHeight(DBE da)

Operation

Returns the number of lines of text displayed in the text editing box.
addToolTip

Declaration

```dxl
void addToolTip(DBE canvas,
       int xpos,
       int ypos,
       int activeWidth,
       int activeHeight,
       type userData
       string toolTipCallback(DBE, type))
```

Operation

Adds a tool tip to the area of a canvas defined by `xpos`, `ypos`, `activeWidth` and `activeHeight`. The upper left corner of the rectangle is defined by `xpos` and `ypos`.

When the canvas is displayed, if the user places the cursor over this rectangle, the callback function is called with `canvas` as the first argument and the `userData` specified in the call to `addToolTip` as the second argument. The `userData` argument can be of any type. The `toolTipCallback` function returns a string, which is displayed at the cursor's position as a tool tip.

You can use `userData` to customize the tool tip message, so that a single callback function can display different messages depending on the area from which it was activated.

Example

This example produces a tool tip, which appears as: The cursor is in the [upper|lower] [left|right] corner.

```dxl
DB box = create "Tooltip example"
string toolTipCallback(DBE xx, string mystring) {
   return "The cursor is in the " mystring " corner"
}
void repaint(DBE c) {
   clearToolTips c
   addToolTip(c, 0, 0, 100, 100, "upper left", toolTipCallback)
   addToolTip(c, 0, 100, 100, 100, "lower left", toolTipCallback)
   addToolTip(c, 100, 0, 100, 100, "upper right", toolTipCallback)
   addToolTip(c, 100, 100, 100, 100, "lower right", toolTipCallback)
}
DBE canvasWithTips = canvas(box, 200, 200, repaint)
```
clearToolTips

Declaration

\texttt{void clearToolTips(DBE canvas)}

Operation

Removes all tool tips associated with \texttt{canvas}.

hasHeader

Declaration

\texttt{void hasHeader(DBE da,}\n\texttt{void cb(DBE el, headerEvent, int hIndex, int param))}

Operation

This function prepares a canvas for headers. It sets a canvas to have a header bar, and defines a callback. This callback is called by one of the four possible event types, through one of the following constants:

- \texttt{headerResize} a header has been resized; \texttt{hIndex} specifies which heading was changed, and \texttt{param} is its new width
- \texttt{headerEdit} header \texttt{hIndex} was double-clicked to request an edit operation
- \texttt{headerSelect} header \texttt{hIndex} was single-clicked to select
- \texttt{headerReorder} header \texttt{hIndex} was dragged into position \texttt{param}

headerAddColumn

Declaration

\texttt{void headerAddColumn(DBE da, string title, int width)}

Operation

Adds a header, with the specified title and width. If there is no header selected, the new column appears at the right of the header bar; otherwise it appears to the left of the currently selected header.

headerChange

Declaration

\texttt{void headerChange(DBE da, int index, string title, int width)}
Operation
Changes the title and width of the header specified by index.

---

**headerRemoveColumn**

**Declaration**

```c
void headerRemoveColumn(DBE da, int index)
```

**Operation**

Deletes the header specified by index from the header bar.

---

**headerReset**

**Declaration**

```c
void headerReset(DBE da)
```

**Operation**

Removes all the headers defined for the canvas, typically before adding new ones.

---

**headerSelect**

**Declaration**

```c
void headerSelect(DBE da, int index)
```

**Operation**

Sets header index to be selected.

---

**headerSetHighlight**

**Declaration**

```c
void headerSetHighlight(DBE da, int index, int highlight)
```

**Operation**

Sets highlight highlight in header index. Valid highlight indices are 0 and 1; 0 is the upper indicator, 1 is the lower indicator.
headerShow

**Declaration**

```c
void headerShow(DBE da, bool onOff)
```

**Operation**

Turns header display on or off in the canvas. Headers must already have been enabled for them to be displayed.

hasScrollbars

**Declaration**

```c
void hasScrollbars(DBE da,
    void cb(DBE canv,
        ScrollEvent Event,
        ScrollSide scrollBar,
        int newPos,
        int oldPos))
```

**Operation**

This function prepares a canvas for scroll bars. It sets a canvas to have scroll bars, and defines a callback. The callback is called with one of the seven possible event types defined through the following constants:

- `scrollToEnd` The thumb has been dragged to the bottom or right-hand end of the bar.
- `scrollToHome` The thumb has been dragged to the top or left-hand end of the bar.
- `scrollPageUp` The user has clicked in the trough above or to the left of the thumb.
- `scrollPageDown` The user has clicked in the trough below or to the right of the thumb.
- `scrollUp` The user has clicked on the left, or up button section of the scroll bar.
- `scrollDown` The user has clicked on the right, or down button section of the scroll bar.
- `scrollToPos` The user has dragged the scroll bar to a new position using the thumb.

In each case, the arguments passed to the callback function are as follows:

- `canv` The canvas to which the event applies.
- `event` One of the scroll events above.
scrollSet

**Declaration**

```c
void scrollSet(DBE da, scrollBar, int maxPos, int view, int pos)
```

**Operation**

This function sets the position and size of the thumb; `maxPos` is the maximum possible position, `view` is the size of the thumb, and `pos` is the position to which the start of the thumb is to be moved. When the thumb is at `maxPos`, the end of the thumb is at `maxPos+view`, making the length of the scroll bar `maxPos+view`. The `scrollBar` argument can be vertical or horizontal.

**Example**

In this example, the thumb has a size of 1 and can move between 0 and 3. The total length of the scroll bar is 4.

```c
scrollSet(can, horizontal, 3, 1, 0)
```

menuBar

**Declaration**

```c
DBE menuBar(DB box)
```

**Operation**

Creates a menu bar within a dialog box. The menu bar automatically appears at the top of the window. Returns a DBE, which must be used for adding menus to the menu bar.

statusBar

**Declaration**

```c
DBE statusBar(DB box,
    string initial,
    int sectionEndPoints[ ],
    [,int no0fSections])
```

**Operation**

This function creates a status bar within a dialog box. The status bar automatically appears at the bottom of the window. The returned DBE must be used for displaying status values.
Status bars contain a number of text areas, which are specified by their end point in pixels. To create a status bar with three areas of 100, 120, and 150 pixels, you must specify sectionEndPoints as:

{100, 220, 370}

You can opt to have either a fixed-size array, and omit noOfSections, or a dynamically filled array, in which case specify the number of sections in noOfSections.

When creating status bars, you should ensure that any dialog box button elements are hidden.

To place a message in the status bar, use the set (status bar message) function.

---

**Menus, status bar and tool bars example**

```dxl
#include <utils/icons.inc>
nt backColor = colorYellow
int sizes[] = {150, 300}
DB menuDemo = create "Menu Demo"
void doClose(DB x) {
    hide x
}
close(menuDemo, false, doClose)
BE mb = menuBar(menuDemo)
DBE sb = statusBar(menuDemo, "Initial", sizes)
void repaint(DBE c) {
    background(c, backColor)
}
DBE canv = canvas(menuDemo, 300, 200, repaint)
string entries[] = {"Size",
                   ">Small",
                   ">Normal",
                   ">Large",
                   
                   
                   
                   
                   
                   
                   
                   
                   
                   
                   
                   
                   
                   
                   
                   
                   "Style",
                   ">Bold",
                   ">Italic"}
char mn[] = {'S',
             'm',
             'N',
             'L',
             't',
             'B',
             'I'}
```
har hot[] = {ddbNone, 
ddbNone, 
ddbNone, 
ddbNone, 
ddbNone, 
ddbNone, 
ddbNone, 

string help[] = {
    "Set size",
    "Small fonts",
    "Normal fonts",
    "Large fonts",
    "Set style",
    "Bold font",
    "Italic font"}

string inactive[] = {
    "Never",
    "Never",
    "Never",
    "Never",
    "Never",
    "Never",
    "Never",
    "Never"}

Sensitivity sensitive(int index) {
    if (index == 2 || index == 6)
        return ddbChecked
    return ddbAvailable
}

void cb(int index) {
    ack "Menu " help[index] " activated"
}

addMenu(mb, "Format", 'F', entries, mn, hot, 
    help, inactive, sensitive, cb)

show menuDemo
This generates the following dialog box:

![Dialog Box Image]

---

**Toolbars**

This section defines functions for using toolbars in dialog boxes, module windows and canvases.

---

**toolBar**

---

**Declaration**

```dxl
toolBar( DB Box
   [, string name,
   Sensitivity mappingCallback()
   , ToolType types[],
   int param[],
   string toolTip[],
   string help[],
   string inactiveHelp[
   [, int noOfTools],
   Sensitivity sensitive(int entryIndex),
   void callback(int entryIndex)
   [, bool newRow,
   bool showName])
```

---

**Operation**

Creates a tool bar within a dialog box. Tool bars can be displayed anywhere in a dialog box and can be placed either with constraints or with the normal automatic placement. Normally tool bars appear between a menu bar and a canvas, which is usually followed by a status bar.
ToolBar contents are specified as arrays all containing the same number of elements. To use fixed-size arrays all containing the same number of elements, omit noOfTools. To use freely-defined arrays, specify the minimum number of elements in noOfTools.

The arguments passed to the function are defined as follows:

- **types**
  - the type of the tool, which can have one of the following values:
    - toolButton: A regular click-to-activate icon.
    - toolToggle: A toggle in/out icon.
    - toolRadio: A mutually exclusive toggle icon.
    - toolCombo: A drop-down list of strings.
    - toolSpacer: A larger gap.
    - toolEditableCombo: A drop down list of strings plus an area in which to enter new strings.

- **param**
  - For a toolButton or toolToggle, this is the id of the icon; the include file utils/icons.inc defines all the icons available as constants; for a toolCombo, it specifies the width of the drop-down list in pixels; there is no value for toolSpacer.

- **toolTip**
  - String that is displayed in the tool tip for this tool.

- **help**
  - String that is displayed in the status bar of the window, if one exists, as the user passes the mouse over an active tool.

- **inactive**
  - String that is displayed in the status bar of the dialog box, if one exists, as the user passes the mouse over an inactive tool.

- **Help**
  - String that is displayed in the status bar of the window, if one exists, as the user passes the mouse over an active tool.

Two callback functions are required: one to determine whether tools are sensitive, and one that is called when a tool is activated. sensitive(int entryIndex) is called for each entry when the toolbar is first displayed or when the updateToolBars function is called.

The function must return one of the following values:

- **ddbUnavailable**
  - The tool\toolbar is unavailable.

- **ddbAvailable**
  - The tool\toolbar is active.

- **ddbChecked**
  - The tool\toolbar is active and has a check beside it.

- **ddbInvisible**
  - The tool\toolbar is not shown

When the user selects an entry, callback(int entryIndex) is called with the index of the tool, and your program must perform the appropriate operation. For both sensitive and callback functions, entryIndex starts at 0, and counts up, so there is a direct correspondence between the array elements and the index returned by the menu.

Name and callback function parameters are optional, as well as booleans determining whether the toolbar is allocated a new row in the container, and whether the name is shown.
If the `name` parameter is specified, the toolbar will be hosted within a container control at the top of the dialog, if not, the toolbar will be generated on the canvas. If `name` is specified then `newRow` and `showName` are mandatory.

The callback function determines how the toolbar option will appear in the context menu for the container control. The possible return values are the same Sensitivity values listed in the table above.

There is also another ToolType available: ToolEditableCombo. It behaves the same as ToolCombo, except the text in the editable area of the combo box is editable, for example a drop-down list of strings plus an area to enter new strings into.

### updateToolBars

**Declaration**

```dxl
void updateToolBars(DB box)
```

**Operation**

Refreshes the state of the tools in all tool bars in dialog box `box`.

### toolBarComboGetSelection

**Declaration**

```dxl
{string|int} toolBarComboGetSelection(DBE tb, int index)
```

**Operation**

Returns the string value of the currently selected option or the index of the currently selected option. The `index` argument specifies which tool is to be processed, counting from 0. All tools are included in the count.

### toolBarComboGetItem

**Declaration**

```dxl
string toolBarComboGetItem(DBE tb, int cIndex, int iIndex)
```

**Operation**

Returns the string value of option `iIndex` in the tool bar combo box specified by `cIndex`. The `iIndex` argument specifies which tool is to be processed, counting from 0. All tools are included in the count.

### toolBarComboSelect

**Declaration**

```dxl
void toolBarComboSelect(DBE tb, int index, {string item|int position})
```
Operation

Selects the option with value \textit{item} (or in indexed \textit{position}) in the tool bar combo box. The \textit{index} argument specifies which tool is to be processed, counting from 0.

\textbf{toolBarComboCount}

\textbf{Declaration}

\begin{verbatim}
int toolBarComboCount(DBE tb, int index)
\end{verbatim}

\textbf{Operation}

Returns the number of options in the tool bar combo box. The \textit{index} argument specifies which tool is to be processed, counting from 0.

\textbf{toolBarComboEmpty}

\textbf{Declaration}

\begin{verbatim}
void toolBarComboEmpty(DBE tb, int index)
\end{verbatim}

\textbf{Operation}

Deletes all the options in the tool bar combo box. The \textit{index} argument specifies which tool is to be processed, counting from 0.

\textbf{toolBarComboAdd}

\textbf{Declaration}

\begin{verbatim}
void toolBarComboAdd(DBE tb, int index, string item)
\end{verbatim}

\textbf{Operation}

Adds an option with value \textit{item} at the end of the tool bar combo box. The \textit{index} argument specifies which tool is to be processed, counting from 0.

\textbf{toolBarComboInsert}

\textbf{Declaration}

\begin{verbatim}
void toolBarComboInsert(DB tb, int index, int position, string item)
\end{verbatim}

\textbf{Operation}

Adds an option with value \textit{item} at the specified \textit{position} of the tool bar combo box list. If the \textit{position} parameter is -1, the item is added to the end of the list. The \textit{index} argument specifies which tool is to be processed, counting from 0.
toolBarComboDelete

Declaration
void toolBarComboDelete(DB tb, int index, int position)

Operation
Used to delete a record within a drop down combo. Takes the position in the list of the item to be deleted. The first item has an index of 0. The index argument specifies which tool is to be processed, counting from 0.

toolBarVisible

Declaration
bool toolBarVisible(Module mod|DBE toolbar|DB box, string name)

Operation
Used to retrieve the visibility state of a toolbar. Only applies to toolbars that are hosted within the appropriate container control (those that were created with the name parameter specified).

toolBarMove

Declaration
void toolBarMove(Module mod|DBE toolbar|DB box, string name, int iposition,
    bool newRow)

Operation
Used to change the position of a toolbar. The toolbar is identified differently depending on which parameters are supplied. This method applies only to toolbars that are hosted within the appropriate container control (those that were created with the name parameter specified).

The newRow parameter defines whether the toolbar is shown on a new row within the ReBar control or not.

toolBarShow

Declaration
void toolBarShow(Module mod|DBE toolbar|DB box, string name, bool bShow)

Operation
Used to change the visibility of a toolbar, as identified by the supplied parameters. Applies only to toolbars that are hosted within the appropriate container control, (those that were created with the name parameter specified).
createEditableCombo

Declaration

void createEditableCombo({linksetCombo|viewCombo|helpCombo})

Operation

Creates an editable combo box in a tool bar in a module or user-created dialog box

toolBarComboGetEditBoxSelection

Declaration

string toolBarComboGetEditBoxSelection(DBE toolbar, int index)

Operation

Returns the selected text from the editable combo box in toolbar where index is the combo box index.

toolBarComboCutCopySelectedText

Declaration

void toolBarComboCutCopySelectedText(DBE toolbar, int index, bool cut)

Operation

Cuts, or copies, the selected text in the editable combo box in toolbar at location index. If cut is true, the selected text is cut to the clipboard. Otherwise, it is copied.

toolBarComboPasteText

Declaration

void toolBarComboPasteText(DBE toolbar, int index)

Operation

Pastes text from the clipboard into the combo box located at index in toolbar. Replaces selected text if there is any.

Colors

This section defines constants and a function that allow you to use color in dialog boxes within Rational DOORS. Colors can be used with attribute types and with canvas dialog box elements.
Logical colors

Logical colors are defined on the options menu.

Declaration

const int color

where color can be one of the following:
logicalCurrentObjectOutline
logicalGridLines
logicalDefaultColor
logicalPageBackgroundColor
logicalTextBackgroundColor
logicalCurrentBackgroundColor
logicalCurrentCellBackgroundColor
logicalTitleBackgroundColor
logicalReadOnlyTextBackgroundColor
logicalUnlockedTextBackgroundColor
logicalDataTextColor
logicalTitleTextColor
logicalSelectedTextColor
logicalReadOnlyTextColor
logicalDeletedTextColor
logicalHighIndicatorColor
logicalMediumIndicatorColor
logicalLowIndicatorColor
logicalGraphicsBackgroundColor
logicalGraphicsShadeColor
logicalGraphicsElideBoxColor
logicalGraphicsTextColor
logicalGraphicsBoxColor
logicalGraphicsLinkColor
logicalGraphicsCurrentColor
logicalGraphicsSelectedColor
logicalGraphicsBoxEdgeColor
logicalLinkPageBackgroundColor
logicalLinkTextBackgroundColor
logicalLinkCurrentBackgroundColor
logicalLinkTitleBackgroundColor
logicalLinkDataTextColor
logicalUser1Color
logicalUser2Color
logicalUser3Color
logicalUser4Color
logicalUser5Color
logicalPageBackgroundColorFilterColor
logicalPageBackgroundColorSortColor
logicalPageBackgroundColorFilterSortColor
logicalTitleBackgroundColor
logicalInPlaceTextColor
logicalInPlaceBackgroundColor
logicalPartitionOutTextColor
logicalPartitionInReadTextColor
logicalPartitionInWriteTextColor
logicalHighlightURLColor
logicalLinksOutIndicatorColor
logicalLinksInIndicatorColor
logical0IndicatorColor
logical11IndicatorColor
logical22IndicatorColor
logical33IndicatorColor
logical44IndicatorColor
logical55IndicatorColor
logical66IndicatorColor
logical77IndicatorColor
logical88IndicatorColor
logical100IndicatorColor
logicalPrintPreviewBackgroundColor
logicalPrintPreviewPageColor
logicalPrintPreviewTextColor
logicalPrintPreviewShadeColor

Actual colors

Actual colors are dependent on the default Rational DOORS setup. These might not make sense if you change your color options.

Declaration

const int color

where color can be one of:

colorLightBlue
colorMediumLightBlue
colorDarkTurquoise
colorPink
colorBlue
colorMaroon
colorRed
colorYellow
colorGreen
colorMagenta
colorCyan
colorWhite
colorOrange
colorBrown
colorBlack
colorGrey82
colorGrey77
colorRedGrey
colorGrey

Real colors

Real colors are the colors you assign to logical colors.

Declaration

const int color
where `color` can be one of:

```c
int realColor_Light_Blue2
int realColor_Light_Blue
int realColor_Dark_Turquoise
int realColor_Pink
int realColor_Blue
int realColor_Maroon
int realColor_Red
int realColor_Yellow
int realColor_Green
int realColor_Cyan
int realColor_Magenta
int realColor_White
int realColor_Orange
int realColor_Brown
int realColor_Purple
int realColor_Navy
int realColor_Sea_Green
int realColor_Lime_Green
int realColor_Rosy_Brown
int realColor_Peru
int realColor_Red_Grey
int realColor_Firebrick
int realColor_Thistle
int realColor_Grey82
int realColor_Grey77
int realColor_Grey66
int realColor_Grey55
int realColor_Grey44
int realColor_Grey33
int realColor_Grey22
int realColor_Grey11
int realColor_Black
int realColor_Ne\\wGrey1
```
int realColor_NewGrey2
int realColor_NewGrey3
int realColor_NewGrey4

Real colors are applied using the realBackground and realColor functions.

getLogicalColorName

Declaration
string getLogicalColorName(int logicalColor)

Operation
Returns the name of logicalColor, which can be any of the values defined in “Logical colors,” on page 591.

getRealColor

Declaration
int getRealColor(int logicalColor)

Operation
Returns the actual color value assigned to logicalColor, which can be any of the values defined in “Real colors,” on page 593.

getRealColorIcon

Declaration
Icon getRealColorIcon(int realColorIndex)

Operation
Returns the icon of realColorIndex, which can be any of the values defined in “Real colors,” on page 593. The icon is for use in a list view or tree view that is all the specified color.

getRealColorName

Declaration
string getRealColorName(int realColor)

Operation
Returns the name of realColor, which can be any of the values defined in “Real colors,” on page 593.
setRealColor

Declaration

```c
string setRealColor(int logicalColor,
                    int realColor)
```

Operation

Sets `logicalColor` (which can have any of the values defined in “Logical colors,” on page 591) to `realColor` (which can be any of the values defined in “Real colors,” on page 593).

Example

This example sets the logical data text color to green:

```c
setRealColor(logicalDataTextColor,
             realColor_Green)
```

Simple placement

This section defines the simple, more or less automatic placement mechanism. This enables you to specify where to place the next element, relative to the previous one. A fully constrainable mechanism is described in “Constrained placement,” on page 599.

beside

Declaration

```c
void beside(DB box)
```

Operation

Places the next element to the right of the last one.

below(element)

Declaration

```c
void below(DB box)
```

Operation

Places the next element below the last one, and aligned with it.
left

Declaration
void left(DB box)
void flushLeft(DB box)

Operation
Places the next element below the last one, at the left of the dialog box.
The flushLeft function is only provided for v2.1 compatibility.

leftAligned

Declaration
void leftAligned(DB box)

Operation
Places the next element in the column at the left-hand side of the dialog box. This is the default placement option: if there are no other alignment options specified items are aligned in a single column.

right

Declaration
void right(DB box)
void flushRight(DB box)

Operation
Places the next element below the last one, at the right of the dialog box.
The flushRight function is only provided for v2.1 compatibility.

opposite

Declaration
void opposite(DB box)

Operation
Places the next element on the same row as the last one, but aligned with the right of the dialog box. After creating the next element another placement mode must be set.
full

Declaration

void full(DB box)

Operation

Specifies that subsequent elements are created at full window width. Labels are aligned on the left; the data area is stretched to be aligned to the right edge of the window. When the dialog box is resized, the element resizes with it. This is most useful with field elements.

stacked

Declaration

void stacked(DBE element)

Operation

Stacks this dialog-box element on top of the preceding one. This is most useful when building an attribute value editor dialog box. Obviously it does not make sense to leave several stacked elements visible, so this is normally used in conjunction with hide.

Example

string enums[] = {"one", "two", "three"}
DB dbBox = "Stacked Example"
DBE stringEdit = field(dbBox, "String:", null, 20, false)
DBE enumEdit = choice(dbBox, "Enum:", enums, 3, 0)
stacked enumEdit
hide enumEdit
DBE intEdit = slider(dbBox, "Int:", 0, 100, 0)
stacked intEdit
hide intEdit
show(dbBox)
Constrained placement

This section defines the constrained placement functions, which expose the full power of the Rational DOORS dialog placement mechanism. As with all power, responsibility is required. You can easily create broken dialog boxes with this mechanism. It is intended for experienced users only, especially those who are sufficiently familiar with simple placement to have reached its limitations.

Constrained placement basics

Here is a standard dialog box, or form:

Dialog box elements are attached to each other and to the dialog box on all edges:

Attachments can be any of the following:
- spaced
- flush
- unattached
- aligned
- inside (normally used within frames or tab strips)
When you place your first element in the dialog box, it has its attachment points connected up for you as follows:

![Diagram of first element connection points]

When you add your next DBE, it is hooked up as follows:

![Diagram of second element connection points]
And so on, until the last one is hooked up to the separator as follows:

Some elements, such as lists, texts, and canvases, come joined onto the form on both vertical edges:

All the other options are implemented in the same style. In implementing a constraint based dialog box layout, it is advisable to draw all the items and their relationships on a piece of paper before encoding them.

**Attachment placement**

The `->` operator is used in constrained placement, as shown in the following syntax:

```
DBE elem -> string side -> string attachment [-> DBE other]
```

where:

- `elem` is a dialog box element of type DBE.
The three operators together make a complete specification for the attachment.

**Note:** You must place dialog box elements in the order they are defined. In the following examples, `mylist` must be declared before `otherList`, and `theFrame` must be declared before `theTab` for the placement to work.

**Example**

```dxl
mylist->"left"->"unattached"
mylist->"right"->"form"
mylist->"left"->"flush"  // not complete
mylist->"left"->"flush"->otherList
theFrame->"left"->"inside"->theTab
theFrame->"right"->"inside"->theTab
theFrame->"top"->"inside"->theTab
theFrame->"bottom"->"inside"->theTab
```
This is a worked example of placing three lists side by side in a dialog box. The first step is to work out the attachments:

Initially, declare the dialog box and lists:

```dxl
DB threeListBox = create "The Three Lists play Carnegie Hall"
DBE list1 = list(threeListBox, "One", 100, 10, listOneData)
DBE list2 = list(threeListBox, "Two", 100, 10, listTwoData)
DBE list3 = list(threeListBox, "Three", 100, 10, listThreeData)
```

Now connect each one up, remembering to disconnect attachments where they would be problematic. This disconnects the first list from the right-hand edge of the form:

```dxl
list1->"right"->"unattached"
```

This connects the left edge of List 2 to the right hand edge of List 1, then aligns the top of List 2 with the top of List 1, and then disconnects the right-hand edge of List 2 from the form:

```dxl
list2->"left"->"flush"->list1
list2->"top"->"aligned"->list1
list2->"right"->"unattached"
```

This does much the same, but note that List 3 remains connected to the form:

```dxl
list3->"left"->"flush"->list2
list3->"top"->"aligned"->list1
```

Instead of aligning the tops of the lists you could connect them all to the form:

```dxl
list1->"top"->"form"
```
but that only works in this case. If List 1 is preceded by another element, for example a field, and you still want the three parallel lists, you need to use alignment.

---

Constrained placement full example program

```dxl
// parallel list DB example
/
  example of DXL dialog boxes which
  builds parallel lists.
*/

DB plistBox = create "Parallel lists"
string listOne[] = {"One", "Two", "Three"}
string listTwo[] = {"Un", "Deux", "Trois"}
string listThree[] = {"Uno", "Dos", "Tres"}
DBE l1 = list(plistBox, "English", 80, 5, listOne)
DBE l2 = list(plistBox, "French", 80, 5, listTwo)
DBE l3 = list(plistBox, "Spanish", 80, 5, listThree)
DBE t1 = text(plistBox, null, null, 80, 50, false)
DBE t2 = text(plistBox, null, null, 80, 50, false)
DBE t3 = text(plistBox, null, null, 80, 50, false)

l1->"right"->"unattached"
l1->"left"->"spaced"->l1
l1->"top"->"aligned"->l1
l2->"right"->"unattached"
l2->"left"->"spaced"->l2
l2->"top"->"aligned"->l1
l3->"right"->"form"
l3->"top"->"spaced"->l1
l3->"right"->"unattached"
l2->"left"->"spaced"->t1
```
void listSel(DBE l) {
    string w = get l
    DBE t
    if (!null w) {
        if (l == l1) t = t1
        if (l == l2) t = t2
        if (l == l3) t = t3
        set(t, w)
    }
}
set(l1, listSel)
set(l2, listSel)
set(l3, listSel)
show plistBox

---

### Progress bar

This section defines functions for the progress bar, which is not a dialog box element as such, but a secondary window that is displayed over the parent window. It contains a title, a message, a progress bar and a **Cancel** button.

---

### progressStart

**Declaration**

```dxl
define progressStart (DB box, string title, string message, int limit)
```

**Operation**

Displays the progress bar and window. The `limit` argument specifies the maximum value that is passed to `progressStep`.
progressStartDisableCancel

**Declaration**

```c
void progressStartDisableCancel(DB box,
        string title,
        string message,
        int limit)
```

**Operation**

The same as the `progressStart` but does not show the Cancel button.

progressStep

**Declaration**

```c
void progressStep(int position)
```

**Operation**

Moves the progress bar to the given position, which must be between 0 and `limit` defined in the preceding call to `progressStart`. Progress can be forward, backward, or cyclic.

progressMessage

**Declaration**

```c
void progressMessage(string message)
```

**Operation**

Sets the message field in the progress window.

progressRange

**Declaration**

```c
void progressRange(string message,
        int position,
        int limit)
```

**Operation**

Specifies new message, position and limit values for the progress bar.
**progressCancelled**

**Declaration**

```dxl
bool progressCancelled()
```

**Operation**

Returns `true` if the Cancel button has been clicked in the progress bar window; otherwise, returns `false`. This can be used to terminate a long-running operation.

---

**progressStop**

**Declaration**

```dxl
void progressStop()
```

**Operation**

Removes the progress bar window from the screen.

---

**Progress bar example**

```dxl
void progCB(DB x) {
    Object o
    int nos = 0
    for o in current Module do nos++
    progressStart(x, "Experiment", "Something",
                   nos)
    nos = 0
    for o in current Module do {
        string h = o."Object Heading"
        progressStep ++nos
        if (null h) h = "null heading"
        progressMessage h
        if (progressCancelled) {
            if (confirm("Exit loop?")) {
                progressStop
                halt
            }
        }
    }
    progressStop
}
```
if (null current Module) {
    ack "Please run from a module"
    halt
}

DB progressDB = create "Progress test"
label(progressDB, "Demonstration of the progress bar")
apply(progressDB, progCB)
show progressDB

### DBE resizing

#### setExtraWidthShare(DBE)

**Declaration**

```dxl
string setExtraWidthShare(DBE control, real share)
```

**Operation**

Sets the share of any extra width that will go to the DBE when the DB is resized.

*share* should be between 0 and 1.0.

#### setExtraHeightShare(DBE)

**Declaration**

```dxl
string setExtraHeightShare(DBE control, real share)
```

**Operation**

Sets the share of any extra height that will go to the DBE *control* when the DB is resized.

*share* should be between 0 and 1.0.

**Example**

```dxl
DB test = create("Field Test")
DBE rich = richText(test, "Rich Text", "This one should expand", 200, 50, false)
DBE readOnlyRich = richText(test, "Rich Text", "This one should expand", 200, 150, true)
DBE rtfField = richField(test, "Rich Field", "This one should be fixed height", 31, false)
```
DBE lab = label(test, "A label")
realize(test)
setExtraHeightShare(rich, 0.25)
setExtraHeightShare(readOnlyRich, 0.75)
show test

---

### HTML Control

The section describes the DXL support for the HTML control.

**Note:** Some of the functions listed below take an ID string parameter to identify either a frame or an HTML element. In each of these methods, frames or elements nested within other frames are identified by concatenating the frame IDs and element IDs as follows: `<top frame ID>/<sub frame ID>/...<element ID>`.

In methods requiring a frame ID, passing `null` into this parameter denotes the top level document. These methods refer to all frame types including IFRAME and FRAME elements.

### htmlView

**Declaration**

```
DBE htmlView(DB parentDB, int width, int height, string URL, bool before_navigate_cb(DBE element, string URL, string frame, string postData), void document_complete_cb(DBE element, string URL), bool navigate_error_cb(DBE element, string URL, string frame, int statusCode), void progress_cb(DBE element, int percentage))
```

**Operation**

Creates an HTML view control where the arguments are defined as follows:

- **parentDB**
  - **The dialog box containing the control.**

- **width**
  - The initial width of the control.

- **height**
  - The initial height of the control.

- **URL**
  - The address that will be initially loaded into the control. Can be `null` to load a blank page (about:blank).
parentDB

before_navigate_cb Fires for each document/frame before the HTML window/frame navigates to a specified URL. It could be used, amongst other things, to intercept and process the URL prior to navigation, taking some action and possibly also navigating to a new URL.

The return value determines whether to cancel the navigation. Returning false cancels the navigation.

Its arguments are defined as follows:
- element: The HTML control itself
- URL: The address about to be navigated to.
- frame: The frame for which the navigation is about to take place.
- postData: The data about to be sent to the server if the HTTP POST transaction is being used.

document_complete_cb Fires for each document/frame once they are completely loaded and initialized. It could be used to start functionality required after all the data has been received and is about to be rendered, for example, parsing the HTML document.

Its arguments are defined as follows:
- element: The HTML control itself
- URL: The loaded address.

navigate_error_cb Fires when an error occurs during navigation. Could be used, for example, to display a default document when internet connectivity is not available.

The return value determines whether to cancel the navigation. Returning false cancels the navigation.

Its arguments are defined as follows:
- elements: The HTML control itself.
- URL: The address for which navigation failed.
- frame: The frame for which the navigation failed.
- statusCode: Standard HTML error code.

progress_cb Used to notify about the navigation progress, which is supplied as a percentage.

set(html callback)

Declaration

void set(DBE HTMLView, bool event_cb(DBE element, string ID, string tag, string event_type))
**Operation**

Attaches a callback to HTML control element that receives general HTML events. The ID argument identifies the element that sourced the event, the tag argument identifies the type of element that sourced the event, and the event_type argument identifies the event type. Note that the only event types currently supported are click and dblclick.

If this function is used with an incorrect DBE type, a DXL run-time error occurs.

---

**set(html URL)**

**Declaration**

```c
void set(DBE HTMLView, string URL)
```

**Operation**

Navigates the given HTMLView to the given URL.

Can only be used to navigate the top level document and cannot be used to navigate nested frame elements.

---

**setURL**

**Declaration**

```c
void setURL(DBE HTMLView, string ID, string URL)
```

**Operation**

Navigates the frame identified by ID to the given URL. The ID might be null.

---

**getURL**

**Declaration**

```c
string getURL(DBE HTMLView, string ID)
```

**Operation**

Returns the URL for the currently displayed frame as identified by its ID. The ID might be null.

---

**get(HTML view)**

**Declaration**

```c
string get(DBE HTMLView)
```

**Operation**

Returns the URL currently displayed in the given HTMLView, if there is one.
get(HTML frame)

Declaration
Buffer get(DBE HTMLView, string ID)

Operation
Returns the URL for the currently displayed frame as identified by its ID.

set(HTML view)

Declaration
string set(DBE HTMLView, Buffer HTML)

Operation
Sets the HTML fragment to be rendered inside the <body> tags by the HTML view control directly. This enables the controls HTML to be constructed dynamically and directly rendered.

setHTML

Declaration
string setHTML(DBE HTMLView, string ID, Buffer HTML)

Operation
Sets the HTML fragment to be rendered inside the <body> tags by the HTML view controls frame as identified by ID. This enables the HTML of the given document or frame to be constructed dynamically and directly rendered.

Note: The contents of the frame being modified must be in the same domain as the parent HTML document to be modifiable. A DXL error will be given on failure (for example, if the wrong type of DBE is supplied).

getHTML

Declaration
Buffer getHTML(DBE HTMLView, string ID)

Operation
Returns the currently rendered HTML fragment inside the <body> tags of the document or frame as identified by its ID.
getBuffer

Declaration
Buffer getBuffer(DBE HTMLView)

Operation
Returns the currently rendered HTML.

getInnerText

Declaration
string getInnerText(DBE HTMLView, string ID)

Operation
Returns the text between the start and end tags of the first object with the specified ID.

setInnerText

Declaration
void setInnerText(DBE HTMLView, string ID, string text)

Operation
Sets the text between the start and end tags of the first object with the specified ID.

getInnerHTML

Declaration
string getInnerHTML(DBE HTMLView, string ID)

Operation
Returns the HTML between the start and end tags of the first object with the specified ID.

setInnerHTML

Declaration
void setInnerHTML(DBE HTMLView, string ID, string html)

Operation
Sets the HTML between the start and end tags of the first object with the specified ID.
Note: The innerHTML property is read-only on the col, colGroup, framSet, html, head, style, table, tBody, tFoot, tHead, title, and tr objects.

**getAttribute**

**Declaration**

```java
string getAttribute(DBE element, string ID, string attribute)
```

**Operation**

Retrieves the value for the requested attribute of the first object with the specified value of the ID attribute. If the attribute does not exist, null is returned.

Returns null on success. Returns error string on failure, for example if the wrong type of DBE is passed in.

**setAttribute**

**Declaration**

```java
void setAttribute(DBE element, string ID, string attribute)
```

**Operation**

Sets the value of the requested attribute for the first object with the specified value of the ID attribute. If the attribute does not exist, it is added to the object.

Displays a DXL error on failure, for example if the wrong type of DBE is passed in.

**Example**

```java
DB dlg
DBE htmlCtrl
DBE htmlBtn
DBE html

void onTabSelect(DBE whichTab){
    int selection = get whichTab
}

void onSetHTML(DBE button){
    Buffer b = create
    string s = get(htmlCtrl)
    print s
```
b = s
set(html, b)
delete b
}

void onGetInnerText(DBE button){
    string s = getInnerText(html, "Text")
    confirm(s)
}

void onGetInnerHTML(DBE button){
    string s = getInnerHTML(html, "Text")
    confirm(s)
}

void onGetAttribute(DBE button){
    string s = getAttribute(html, "Text", "Align")
    confirm(s)
}

void onSetInnerText(DBE button){
    Buffer b = create
    string s = get(htmlCtrl)
    setInnerText(html, "Text", s)
}

void onSetInnerHTML(DBE button){
    Buffer b = create
    string s = get(htmlCtrl)
    setInnerHTML(html, "Text", s)
}

void onSetAttribute(DBE button){
    Buffer b = create
string s = getAttribute(html, "Text", "Align")
if (s == "left"){
    s = "center"
}
else if (s == "center"){
    s = "right"
}
else if (s == "right"){
    s = "left"
}
setAttribute(html, "Text", "align", s)
}

bool onHTMLBeforeNavigate(DBE dbe, string URL, string frame, string body){
    string buttons[] = {"OK"}
    string message = "Before navigate - URL: " URL "\nFrame: " frame "\nPostData: " body "\n" print message "" return true
}

void onHTMLDocComplete(DBE dbe, string URL){
    string buttons[] = {"OK"}
    string message = "Document complete - URL: " URL "\n" print message "" string s = get(dbe)
    print "url: " s "\n"
}

bool onHTMLError(DBE dbe, string URL, string frame, int error){
    string buttons[] = {"OK"}
    string message = "Navigate error - URL: " URL "; Frame: " frame "; Error: " error "\n" print message ""
return true
}

void onHTMLProgress(DBE dbe, int percentage){
    string buttons[] = {"OK"}
    string message = "Percentage complete: " percentage "\r\n"
    print message
    return true
}

dlg = create("Test", styleCentered | styleThemed | styleAutoparent)
htmlCtrl = text(dlg, "Field:", "<html><body><p id="Text"><p align="center">Welcome to <b>DOORS <i>ERS</i></p><p><html>", 200, false)
htmlBtn = button(dlg, "Set HTML...", onSetHTML)
DBE getInnerTextBtn = button(dlg, "Get Inner Text...", onGetInnerText)
DBE getInnerHTMLBtn = button(dlg, "Get Inner HTML...", onGetInnerHTML)
DBE getAttributeBtn = button(dlg, "Get Attribute...", onGetAttribute)
DBE setInnerTextBtn = button(dlg, "Set Inner Text...", onSetInnerText)
DBE setInnerHTMLBtn = button(dlg, "Set Inner HTML...", onSetInnerHTML)
DBE setAttributeBtn = button(dlg, "Set Attribute...", onSetAttribute)

DBE frameCtrl =  frame(dlg, "A Frame", 800, 500)

string strTabLabels[] = {"One","Two"}
DBE tab =  tab(dlg, strTabLabels, 800, 500, onTabSelect)

htmlCtrl->"top"->"form"
htmlCtrl->"left"->"form"
htmlCtrl->"right"->"unattached"
htmlCtrl->"bottom"->"unattached"

htmlBtn->"top"->"spaced"->htmlCtrl
htmlBtn->"left"->"form"
htmlBtn->"right"->"unattached"
htmlBtn->"bottom"->"unattached"

getInnerTextBtn->"top"->"spaced"->htmlCtrl
getInnerTextBtn->"left"->"spaced"->htmlBtn
getInnerTextBtn->"right"->"unattached"
getInnerTextBtn->"bottom"->"unattached"

getInnerHTMLBtn->"top"->"spaced"->htmlCtrl
getInnerHTMLBtn->"left"->"spaced"->getInnerTextBtn
getInnerHTMLBtn->"right"->"unattached"
getInnerHTMLBtn->"bottom"->"unattached"

getAddressBtn->"top"->"spaced"->htmlCtrl
getetAddressBtn->"left"->"spaced"->getInnerHTMLBtn
getetAddressBtn->"right"->"unattached"
getetAddressBtn->"bottom"->"unattached"

setInnerTextBtn->"top"->"spaced"->htmlBtn
setInnerTextBtn->"left"->"aligned"->getInnerTextBtn
setInnerTextBtn->"right"->"unattached"
setInnerTextBtn->"bottom"->"unattached"

setInnerHTMLBtn->"top"->"spaced"->htmlBtn
setInnerHTMLBtn->"left"->"spaced"->setInnerTextBtn
setInnerHTMLBtn->"right"->"unattached"
setInnerHTMLBtn->"bottom"->"unattached"

setAttributeBtn->"top"->"spaced"->htmlBtn
setAttributeBtn->"left"->"spaced"->setInnerHTMLBtn
setAttributeBtn->"right"->"unattached"
setAttributeBtn->"bottom"->"unattached"

frameCtrl->"top"->"spaced"->setInnerTextBtn
frameCtrl->"left"->"form"
HTML Edit Control

The section describes the DXL support for the HTML edit control.

The control behaves in many ways like a rich text area for entering formatted text. It encapsulates its own formatting toolbar enabling the user to apply styles and other formatting.

htmlEdit

Declaration

DBE htmlEdit(DB parentDB, string label, int width, int height)

Operation

Creates an HTML editor control inside parentDB.
htmlBuffer

Declaration
Buffer getBuffer(DBE editControl)

Operation
Returns the currently rendered HTML fragment shown in the control. The fragment includes everything inside the <body> element tag.

set(HTML edit)

Declaration
void set(DBE editControl, Buffer HTML)

Operation
Sets the HTML to be rendered by the edit control. The HTML fragment should include everything inside, but not including, the <body> element tag.

Example

DB MyDB = create "hello"
DBE MyHtml = htmlEdit(MyDB, "HTML Editor", 400, 100)

void mycb (DB dlg){

    Buffer b = getBuffer MyHtml
    string s = stringOf b
    ack s
}

apply (MyDB, "GetHTML", mycb)
set (MyHtml, "Initial Text")
show MyDB
Chapter 23
Templates

This chapter describes template functions and expressions:

- Template functions
- Template expressions

Template functions

This section defines functions that allow you to construct a simple, formal module template: essentially a table of contents. The functions use the Template data type. The templates section of the DXL Library contains many examples.

**Note:** If you are creating new DXL files that are to be included in the templates list available in the Rational DOORS client, and the template name, which appears at the top of the DXL file, uses unicode multibyte characters, you must save the DXL file as UTF-8 encoding.

**template**

**Declaration**

Template template(string h)

**Operation**

Returns a template that builds a single object with string h as its heading.

**Example**

Template t = template "trivial"

**instance**

**Declaration**

void instance(Template t)

void instance(below(Template t))

**Operation**

The first form creates an instance of template t immediately after the current object and at the same level, or at the first object position in an empty module.

The second form creates an instance of the template below the current object.
Example

// same level
Template t = template "trivial"
instance t

// below
Template t = template "trivial"
instance below t

Template expressions

This section defines the operators used to assemble templates in expressions.

Operators

Template expression operators can be used as shown in the following syntax:
Template t <> string h
Template t << string h
Template t >> string h

Each operator adds an object with heading h at a specific level of template t, and returns the new template. The levels are:

<> current level
<< next level down
>> next level up

The following syntax can be used to specify a number of levels up:
Template t >> int n <> string h

In this form, the second operator can be replaced by << or >>.

Example

This example adds an object at the same level, then another at the level below:
Template t = template "A" <> "B" << "B.A"
instance t

This example adds an object at the same level, then a series of objects each one level lower. B.A.A.A.A is four levels below A and B; the instance of the new template needs to be at the same level as B, so C is added four levels above B.A.A.A.A:
Template t = template "A" <>
  "B" <<
  "B.A" <<
  "B.A.A" <<
  "B.A.A.A" <<
  "B.A.A.A.A" >> 4 <>
  "C"

instance t

This example is equivalent to:

Template t = template "A" <>
  "B" <<
  "B.A" <<
  "B.A.A" <<
  "B.A.A.A" <<
  "B.A.A.A.A" >> 3 >>
  "C"

instance t
Chapter 24

Rational DOORS window control

This chapter describes the DXL library and Addins menus. It also defines functions and standard items that control the way Rational DOORS displays information and its windows.

- The DXL Library and Addins menus
- Module status bars
- Rational DOORS built-in windows
- Module menus

The DXL Library and Addins menus

DXL libraries are directories stored in the Rational DOORS file tree. They can be found at $DOORSHOME/lib/dxl. Each library must contain a description file for that library with the same name as the directory but with a .hlp extension. Only files ending in .dxl are recognized as library elements.

The standard directory adds functions to the Rational DOORS formal module Tools menu. Each directory in addins appears as a new menu in formal modules. Subdirectories appear as submenus.

The order of menu items as well as their names, mnemonics and accelerators are defined in an index file with the same name as the library directory but with a .idx extension.

As an example, see the user-defined function fn.dxl included in the formal module menus:

dxl/addins/addins.hlp
  addins.idx
  user/user.hlp
    user.idx
    fn.dxl

Library description file format

Each library must contain a description file for the library with the same name as the directory but with a .hlp extension.

The first line of the description file is a one-line description of the library. The rest of the file can expand on this, with descriptive text providing detailed information about the library.

Example

This example is the start of file: $DOORSHOME/dxl/addins/acme/acme.hlp:
The ACME Spindles Inc DXL function library
This library contains a set of functions
developed by ACME Spindles Inc to support
our internal use of Rational DOORS.

Menu index file format

Each subdirectory within the addins directory can contain a menu index file with the same name as the directory but with a .idx extension. Each line of the menu index file must contain:

- DXL file or directory name, without extension
- mnemonic (character used with ALT to access menu from keyboard)
- accelerator (character used with CTRL to access menu from keyboard); an underscore means no accelerator
- menu label

A line containing only hyphens (-) (as in line 3 of the following example), inserts a separator within the menu.

Example

This example is the first four lines of file $DOORSHOME/dxl/addins/acme/acme.idx:

comps C _ Component book
template T _ Templates
---------------------------
parsers I _ Input parsers

Menu DXL file format

Each DXL file to be included in the menu must conform to the following comment convention:

- The first line of the file contains a // comment with a single-line description of the program, which appears in the DXL Library window.

- This must be followed by a /* ... */ multi-line comment which describes in more detail what the program does. This can be viewed from the DXL Library window by clicking the Describe button.

Example

File: $DOORSHOME/dxl/addins/acme/example.dxl

// A simple example program
/*
This program simply displays an ack box.
*/

ack "This is a Menu DXL example program"
Alternative Addins Location

Additional addins directories can also be created outside of the standard Rational DOORS installation path. Here are steps on how to create a such a configuration on a Rational DOORS client machine:

- Create the directory where you want to contain your addins library, which can be created on any drive, for example E:\addins. Each library must contain a description file for the library with the same name as the directory but with a .hlp extension. See above for further details.

- Create another directory for your DXL, for example E:\addins\MyDXL. Again, each directory must contain a description file for the library with the same name as the directory but with a .hlp extension. See above for further details.

- Add your DXL, making sure the comment convention used in ‘Menu DXL file format’ above is adhered to.

- Create a Registry string value for your addins:
  - This is created in the key HKEY_LOCAL_MACHINE\SOFTWARE\Telelogic\DOORS\<DOORS version number>\Config
  - A new string value should be created in this key with ‘Value Name’ set to ‘Addins’ and ‘Value Data’ set to the path of the addins directory, for example E:\addins

Module status bars

This section defines functions for the module window status bar, in which Rational DOORS displays information such as the user name, access rights, or other information. These functions allow your DXL program to place information in the status bar.

status

Declaration

void status(Module m,
            string message)

Operation

Displays string message in the left-most field of the status bar of module m.

Example

status(current Module, "Power validated")
menuStatus

Declaration
void menuStatus(Module m
[,string message])

Operation
Displays string message in the full status bar area of module m, in the same way that help menu explanations are displayed.

If message is omitted, the status bar returns to its normal state.

Example
menuStatus(current Module, "Module exported in
GREN III format")

updateToolBars

Declaration
void updateToolBars(Module m)

Operation
Redraws the tool bars for module m. This might be needed when certain display modes are altered using a DXL program.

Rational DOORS built-in windows

This section gives the syntax for functions that operate on Rational DOORS built-in windows. The functions use an internal data type, so declarations are not stated.

See also “Scrolling functions,” on page 697.

window

Syntax
window m

Operation
Returns a handle to the window displaying module m, for use in the width and height functions.

Example
print width window current Module
show (window)

Syntax
show win

Operation
Shows a Rational DOORS built-in window, if it is available.

hide

Syntax
hide win

Operation
Hides a Rational DOORS built-in window, if it is showing.

Specific windows

Syntax
editor(attrRef)
print(m)

where:

- \( m \) is a module of type Module
- \( attrRef \) is in one of the following formats:
  - \((\text{Object} \ o). (\text{string} \ attrName)\)
  - \((\text{Module} \ m). (\text{string} \ attrName)\)
  - \((\text{Link} \ l). (\text{string} \ attrName)\)

Operation
These functions return the appropriate window, for use with show (window) and hide, as follows:

- \( \text{editor} (attrRef) \) object attribute editor
- \( \text{print} \) print

Example
show editor(current Object)."Status"
show print current Module
Module menus

This section lists constants and gives the syntax for functions that create and manage menus. Many of the functions use internal data types, so declarations are not stated. For examples of how to build menus, look in $DOORSHOME/lib/dxl/config.

Standard menus and submenus

The following constants are defined as standard menus and submenus:

clipCopyMenu
clipPasteMenu
clipboardMenu
projectMenu
moduleMenu
editMenu
oleMenu
viewMenu
objectMenu
linkMenu
linksetMenu
attributeMenu
columnMenu
extractMenu
toolsMenu
usersMenu
optionsMenu
helpMenu
objCopyMenu
objCreateMenu
objMoveMenu
objUnlockMenu

Standard items

The following constants are defined as standard items:
OLECutItem
OLECopyItem
OLEPasteItem
OLEPasteSpecialItem
OLEClearItem
OLEInsertItem
OLERemoveItem
OLEVerbItem
attrDefItem
attrTypeItem
clipCutItem
clipCopyFlatItem
clipCopyHierItem
clipPasteItem
clipPasteDownItem
clipClearItem
columnCreateItem
columnEditItem
columnDeleteItem
columnLeftJustifyItem
columnRightJustifyItem
columnCenterJustifyItem
columnFullJustifyItem
columnUseInGraphicsItem
columnUseAsToolTipsItem
dispGraphicsItem
dispOutlineItem
dispFilterDescendantsItem
dispFilteringItem
dispSortingItem
dispDeletionItem
dispReqOnlyItem
dispFilterParentsItem
dispGraphicsLinksItem
dispGraphicsToolTipsItem
dispLevelAllItem
dispLevel1Item
dispLevel2Item
dispLevel3Item
dispLevel4Item
dispLevel5Item
dispLevel6Item
dispLevel7Item
dispLevel8Item
dispLevel9Item
dispLevel10Item
editDXLItem
editUsersItem
EXIT_Item
extractSetupItem
extractSameItem
extractDownItem
filterItem
helpContentsItem
helpSearchItem
helpIndexItem
helpHelpItem
helpProjManItem
helpFormalItem
helpDescriptiveItem
helpLinkItem
helpAboutItem
inplaceRejectItem
inplaceAcceptItem
inplaceHeadingItem
inplaceTextItem
inplaceAttrItem
inplaceResetAttrItem
linkCreateItem
linkEditItem
linkDeleteItem
linkSourceItem
linkTargetItem
linkMatrixItem
linkGraphicsItem
linksetCreateItem
linksetDeleteItem
linksetRefreshItem
modAccessItem
modAttrEditItem
modBaselineItem
modCloseItem
modHistoryItem
modLayoutItem
modPrintItem
modSaveItem
modDowngradeItem
modPrintPreviewItem
objAccessItem
objCompressItem
objUncompressItem
objCompressionItem
objCopyItem
objCopyDownItem
objCreateItem
objCreateDownItem
objDeleteItem
objUndeleteItem
objPurgeItem
objEditItem
objHistoryItem
objMoveItem
objMoveDownItem
objLockItem
colorOptionsItem
fontOptionsItem
optionsSaveItem
optionsRestoreItem
optionsDefaultsItem
pictureItem
createProjectItem
openProjectItem
deleteProjectItem
undeleteProjectItem
purgeProjectItem
duplicateProjectItem
closeProjectItem
projectAttrItem
unlockModulesItem
purgeModulesItem
projectArchiveItem
projectRestoreItem
createFormalModuleItem
createLinkModuleItem
createDescriptiveModuleItem
openModuleEditItem
openModuleShareItem
openModuleReadItem
deleteModuleItem
undeleteModuleItem
purgeModuleItem
duplicateModuleItem
renameModuleItem
archiveModuleItem
restoreModuleItem
showFormalModulesItem
showLinkModulesItem
showDescriptiveModulesItem
showDeletedModulesItem
showDeletedProjectsItem
sortNameItem
sortTypeItem
sortDescriptionItem
selectItem
deselectItem
sortItem
spellCheckItem
undoItem
redoItem
viewCreateItem
viewShowItem
viewDeleteItem

Standard combo box controls

The following constants are defined as standard combo box controls:

linksetCombo
viewCombo
helpCombo

createMenu

Syntax

createMenu(menuIdentifier
    [,string label,
    char mnemonic,
    string dxlDirectory])

createMenu(int mappingFunction(),
            string label,
            char mnemonic,
            string dxlFile)
Operation

Creates a standard or configurable menu or submenu, according to context. The arguments are defined as follows:

- **menuIdentifier**: Provides a standard menu definition, which is particularly useful as a source of menu gray-out behavior; for a standard menu, it must take one of the values listed in “Standard menus and submenus,” on page 630.

- **label**: The text of the menu item or null.

- **mnemonic**: The character of the label that is to be used with ALT for keyboard access, or null.

- **dxlDirectory**: Provides the name of a standard-format DXL library directory, or null.

- **mappingFunction()**: Callback function which returns an integer that specifies whether the menu item is available, checked, or invisible; possible values are:
  - menuAvailable_
  - menuAvailableChecked_
  - menuUnavailable_
  - menuInvisible_

  **Note:** This functionality is not supported for menus created on the module menu bar. It is just for menus within these menu bar menus that have been created. Mapping functions have to be defined in a file inside ${DOORS_HOME}/lib/dxl/startupfiles and cannot be in the same file as the perms that call them.

- **dxlFile**: Full path name of the DXL file containing the menu.

For a standard menu only the *menuIdentifier* is required. For a configurable menu or submenu, *menuIdentifier* provides basic information, including predefined gray-out behavior. In this case it can also be null. If not null, *label* and *mnemonic* override the predefined appearance of the menu. If not null, the contents of *dxlDirectory* are used for the menu.

The second form takes a DXL mapping function as a callback. If not null, the contents of *dxlFile* are used for the menu.

**Example**

```dxl
createMenu moduleMenu
```
createButtonBar

**Declaration**

```dxl
generic void createButtonBar([string name, sensitivity mappingCallback(), bool newRow, bool showName])
```

**Operation**

Creates a button bar in a module or user-created dialog box. If the name is supplied the toolbar will be hosted in a container control at the top of the dialog, if not it will be generated on the canvas. The `newRow` parameter defines whether the toolbar is shown on a new row within the container control or not. The `showName` parameter defines whether the name of the toolbar is shown or not. Both `newRow` and `showName` are mandatory when the toolbar is hosted outside a canvas.

When a user right-clicks within the container control of a dialog, a context menu will be shown to allow the user to show or hide the toolbars inside it. The mapping callback function is called for each toolbar if provided to allow the DXL to control the display of context menu items for the given toolbar. It can be set to null. The function must return one of the following `Sensitivity` values:

- `ddbUnavailable`: The tool is unavailable.
- `ddbAvailable`: The tool is active.
- `ddbChecked`: The tool is active and has a check beside it.
- `ddbInvisible`: The tool is not shown.

createItem

**Declaration**

```dxl
generic void createItem(standardItem
    [, string label,
    char mnemonic,
    char accelerator,
    {IconID icon_id|string iconFileName},
    string tooltip,
    string helptext,
    string inactiveHelp,
    string dxlFile])
```

```dxl
generic void createItem(int mappingFunction(),
    string label,
    char mnemonic,
    char accelerator,
    int modifierKeyFlags,
    {IconID icon_id|string iconFileName},
```

DDL values:

- `ddbUnavailable`: The tool is unavailable.
- `ddbAvailable`: The tool is active.
- `ddbChecked`: The tool is active and has a check beside it.
- `ddbInvisible`: The tool is not shown.

DXL Reference Manual
string tooltip,
string helptext,
string inactiveHelp,
string dxlFile)

void createItem(int mappingFunction(),
    void dxlCallback(),
    string label,
    char mnemonic,
    char accelerator,
    int modifierKeyFlags,
    {IconID icon_id|string iconFileName},
    string tooltip,
    string helptext,
    string inactiveHelp)

Operation

Creates a DXL menu item in a module or user-created dialog box. In the first form, if the optional arguments are omitted, creates a standard item. The arguments are defined as follows:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>standardItem</td>
<td>Provides a standard menu item definition, which is particularly useful as a source of menu gray-out behavior; it must have one of the values listed in “Standard items,” on page 630.</td>
</tr>
<tr>
<td>label</td>
<td>The text of the menu item.</td>
</tr>
<tr>
<td>mnemonic</td>
<td>The character of the label that is to be used with ALT for keyboard access or null.</td>
</tr>
<tr>
<td>accelerator</td>
<td>The character that is to be used with the CTRL for direct keyboard access or null. This option does not function for pop-up menus.</td>
</tr>
<tr>
<td>modifierKeyFlags</td>
<td>Used in conjunction with the accelerator parameter to change which key should be pressed with the accelerator key. Possible values are modKeyNone, modKeyCtrl, modKeyShift and null.</td>
</tr>
<tr>
<td>icon_id</td>
<td>The icon identifier of the standard icon, used for button bars only or null; it must have one of the values listed below.</td>
</tr>
<tr>
<td>iconFileName</td>
<td>The file to be used as an icon. Must be a valid icon format .ico file.</td>
</tr>
<tr>
<td>tooltip</td>
<td>Text to be displayed in the button-bar tooltip or null.</td>
</tr>
<tr>
<td>helptext</td>
<td>Text to appear in the status bar for the item (if active) or null.</td>
</tr>
<tr>
<td>inactiveHelp</td>
<td>Text to appear in the status bar for the item (if inactive) or null.</td>
</tr>
</tbody>
</table>
dxlFile

Complete path name of the DXL file to execute or null (usually null).

mappingFunction()

Callback function which returns an integer that specifies whether the menu item is available, checked, or invisible; possible values are:

menuAvailable_
menuAvailableChecked_
menuUnavailable_
menuInvisible_

Mapping functions have to be defined in a file inside $DOORSHOME\lib\dxl\startupfiles and cannot be in the same file as the perms that call them.

dxlCallback()

Callback function which runs when the menu is selected (instead of running a DXL file).

The possible values for IconID constants are:

levelAllIcon
level1Icon
level2Icon
level3Icon
level4Icon
level5Icon
level6Icon
level7Icon
level8Icon
level9Icon
level10Icon
dispGraphicsIcon
dispOutlineIcon
dispFilterIcon
dispSortIcon
createObjSameIcon
createObjDownIcon
deleteObjIcon
columnInsertIcon
columnEditIcon
columnRemoveIcon
justifyLeftIcon
justifyRightIcon
justifyCenterIcon
justifyFullIcon
folderOpenIcon
folderNewIcon
folderCloseIcon
projOpenIcon
projNewIcon
projCloseIcon
editUsersIcon
createModIcon
editModIcon
shareModIcon
readModIcon
copyModIcon
deleteModIcon
createLinkIcon
editLinkIcon
deleteLinkIcon
matrixModeIcon
startLinkIcon
endLinkIcon
createLinksetIcon
createFormalModIcon
createLinkModIcon
deleteLinksetIcon
editHeadingIcon
editTextIcon
extractObjIcon
extractOneDownIcon
showMarkedObjsIcon
Example

createItem(linkCreateItem, "Create link", 'C',
        null, null, null, null, null, null, null)
createCombo

Syntax

createCombo({linksetCombo|viewCombo})

Operation

Creates a standard combo box in a toolbar.

Example

createButtonBar
separator
createCombo viewCombo
separator
end buttonbar

createEditableCombo

Syntax

createEditableCombo({linksetCombo|viewCombo})

Operation

Creates an editable combo box in a toolbar.

createPopup

Declaration

void createPopup()

Operation

Creates a popup menu in a module or user-created dialog box.

separator(menu)

Declaration

void separator()

Operation

Adds a menu separator.
end(menu, button bar, popup)

**Syntax**
end({menu|buttonbar|popup})

**Operation**
Ends a menu, button bar or popup section.

**Example**
end menu
end buttonbar
end popup
Chapter 25
Display control

This chapter describes DXL functions that control what information is displayed in Rational DOORS module windows.

- Filters
- Compound filters
- Filtering on multi-valued attributes
- Sorting modules
- Views
- View access controls
- View definitions
- Columns
- Scrolling functions
- Layout DXL

Filters

This section defines operators and functions for building display filters.

The data type Filter enables the construction of complex filters which can then be applied with the set command. The data type LinkFilter can take one of the following values:

- `linkFilterIncoming`
- `linkFilterOutgoing`
- `linkFilterBoth`

These enable the construction of filters with reference to incoming links, outgoing links, or both. They are used with the hasLinks and hasNoLinks functions.

The DXL functions for filtering mimic the capability provided by the Rational DOORS user interface, except for accept and reject, which allow a DXL program to set an arbitrary filter.

**Note:** If you define an advanced filter and specify a rule such as `<dxlAttribute> contains <sometext>`, the filter runs on all objects in the module when you add the rule to the list of rules. If this creates excessive delays in the filter implementation, you can modify this behavior by replacing the `filter_gui.inc` file. With this modified file, the rule is applied only after you click **Apply**. You are then prompted to apply the rule to each module using the Next and Previous buttons. The `filter_gui.inc` file is located in the `\lib\dxl\standard\filter` directory. You can obtain the revised file at the technote [http://www.ibm.com/support/docview.wss?uid=swg21585679](http://www.ibm.com/support/docview.wss?uid=swg21585679). Back up the current `filter_gui.inc` file before replacing it.
**attribute(value)**

This function is used to generate a filter attribute handle, as shown in the following syntax:

```
attribute(string attrName)
```

The returned handle for the attribute named `attrName` is used by other functions.

**Example**

This example filters on all objects in the current module that have a "Priority" attribute value of "Mandatory".

```
set(attribute "Priority" == "Mandatory")
```

**column(value)**

**Syntax**

```
Filter column(string ColumnName, 
    string SearchText [, bool CaseSensitive, 
    bool RegularExpression])
```

**Operation**

Filters on the contents of `ColumnName`. The last two parameters are optional.

**Attribute comparison**

Operators can be used to compare filter attribute handles and text strings.

**Syntax**

The syntax for using these operators is as follows:

```
attribute(string attr) operator string text
```

where:

- `attr` is the name of the attribute
- `operator` is one of `==` `!=` `<` `<=` `>` `>=`
- `text` is a string

**Operation**

Compares the filter attribute handle returned by the call to `attribute` with the string `text`. If `text` is a variable of another type, you can convert it to a string by concatenating it with the empty string.
Example
This example filters on only those objects in the current module that have attribute "Cost" values greater than 4:

```dxl
set(attribute "Cost" > "4")
// using wrong type
real cost = 4.0
set(attribute "Cost" > cost "")
```

accept

**Declaration**

```dxl
void accept(Object o)
```

**Operation**

Marks object `o` as accepted under the current filter. This enables a DXL program to set an arbitrary filter on the current module. Compare with the `reject` function.

addFilter

**Declaration**

```dxl
void addFilter([Module m,]
    Filter f
    int &accepted,
    int &rejected)
```

**Operation**

Adds a filter in the current module, or to module `m` where it is specified. The third and fourth parameters pass back the number of objects accepted and rejected respectively under the filter.

contents

**Declaration**

```dxl
Filter contents(string text[, bool caseSensitive[, bool useRegExp]])
```

**Operation**

Filters on objects that include the string `text` in any string or text attributes. If `caseSensitive` is set to `true`, the filter takes character case into account when searching. If `caseSensitive` is `false`, the filter ignores case. If `caseSensitive` is omitted, the filter accepts regular expressions.

The optional `useRegExp` parameter enables the use of regular expressions to be specified independently of case sensitivity.
Example
This example matches objects that contain literally f.*h, but not F.*H, f.*H, or F.*H.
Filter f = contents("f.*h",true)
This example matches objects that contain the regular expression f.*h, for example, fish or fourteenth.
Filter f = contents "f.*h"

contains

Declaration
Filter contains(attribute(string attributeName), string text, [bool caseSensitive[, bool useRegExp]])

Operation
Filters on objects that include the string text in a specific attribute attributeName. If caseSensitive is set to true, the filter takes character case into account when searching. If caseSensitive is false, the filter ignores case. If caseSensitive is omitted, the filter accepts regular expressions.

Example
Module m = current
Filter f = contains(attribute "Object Text", "shall", false)
set f
filtering on

excludeCurrent

Declaration
Filter excludeCurrent()

Operation
Excludes the current object from the filter.

excludeLeaves

Declaration
Filter excludeLeaves()
Operation
Excludes leaves from the filter.

filterTables

Declaration
void filterTables(bool onOff)
bool filterTables(Module m)

Operation
The first form sets whether tables are filtered in the current module.
The second form returns whether table contents are being filtered in the specified module m.

getSimpleFilterType_

Declaration
int getSimpleFilterType_(Filter)

Operation
Returns the type of the simple filter; attribute, link, object, or column. Please note that the returned value corresponds to the index of the appropriate tab page on the filter dialog. If the specified filter is not a simple filter, -1 is returned.

getAddressFilterSettings_

Declaration
bool getAttributeFilterSettings_(Module, Filter,
                                             string& attributeName,
                                             int& comparisonType,
                                             string& comparisonValue,
                                             bool& matchCase,
                                             bool& useRegexp)

Operation
Returns details of the specified attribute filter in the return parameters. The function returns false if the filter is not a valid attribute filter.
The comparisonType parameter returns the internal index of the comparison. This is different to the index that is used in the associated combo box on the filter dialog. The translation is performed by the DXL code.
getLinkFilterSettings_

Declaration

bool getLinkFilterSettings_(Module,
    Filter,
    bool& mustHave,
    int& linkType,
    string& linkModuleName)

Operation

Returns details of the specified link filter in the return parameters. The function returns false if the filter is not a valid link filter.

The linkType parameter returns a value that maps directly to the appropriate combo box.

The linkModuleName parameter returns an asterisk if links are allowed through any module, or the module name.

ggetObjectFilterSettings_

Declaration

bool getObjectNameFilterSettings_(Module,
    Filter,
    int& objectFilterType)

Operation

Returns details of the specified object filter in the return parameter. The function returns false if the filter is not a valid object filter.

The objectFilterType parameter returns a value that maps directly to the radio group on the dialog.

ggetColumnFilterSettings_

Declaration

bool getColumnFilterSettings_(Module,
    Filter,
    string& columnName,
    string& comparisonValue,
    bool& matchCase,
    bool& useRegExp)
Operation

Returns details of the specified column filter in the return parameters. The function returns false if the filter is not a valid column filter.

includeCurrent

Declaration

Filter includeCurrent()

Operation

Includes the current object in the filter.

includeLeaves

Declaration

Filter includeLeaves()

Operation

Includes leaves in the filter.

hasLinks

Declaration

Filter hasLinks(LinkFilter value,
               string linkModName)

Operation

Includes in the filter objects that have links through link module linkModName. The string can also take the special value "*", which means any link module. The value argument defines the type of links; it can be one of linkFilterIncoming, linkFilterOutgoing, or linkFilterBoth.

Example

This example filters on objects that have incoming links through any link module:

Module m = current
Filter f = hasLinks(linkFilterIncoming, "*")
set(m, f)
filtering on
**hasNoLinks**

**Declaration**

Filter hasNoLinks(LinkFilter value,
                 string modName)

**Operation**

Includes in the filter objects that have no links through link module \(\text{linkModName}\). The string can also take the special value \(\ast\), which means any link module. The \(\text{value}\) argument defines the type of links; it can be one of \(\text{linkFilterIncoming}\), \(\text{linkFilterOutgoing}\), or \(\text{linkFilterBoth}\).

**Example**

This example filters on objects that have neither incoming nor outgoing links through the link module Project Links:

Module \(m\) = current

Filter \(f\) = hasNoLinks(linkFilterBoth,
                      "Project Links")

\(\text{set}(m, f)\)

filtering on

**isNull**

**Declaration**

Filter isNull(attribute(string \(\text{attrName}\))

**Operation**

Returns \(\text{true}\) if the call to \(\text{attribute}\) returns \(\text{null}\).

Returns \(\text{false}\) if the call to \(\text{attribute}\) returns an attribute other than \(\text{null}\).

**notNull**

**Declaration**

Filter notNull(attribute(string \(\text{attrName}\))

**Operation**

Returns \(\text{true}\) if the call to \(\text{attribute}\) returns an attribute other than \(\text{null}\).

Returns \(\text{false}\) if the call to \(\text{attribute}\) returns \(\text{null}\).
reject

Declaration

void reject(Object o)

Operation

Marks object o as rejected under the current filter. This enables a DXL program to set an arbitrary filter on the current module. Compare with the accept function.

Example

Object o
filtering off
// following loop only accesses displayed objects
// cycle through all displayed objects
for o in current Module do {
    bool accepted = false
    Link l
    for l in o->"*" do {
        // accept o if any out going links
        accept o
        accepted = true
        break
    }
    if (!accepted)
    {
        reject o   // no outgoing links, reject o
    }
}  
filtering on             // activate our new filter
set(filter)

**Declaration**

```c
void set(Module m,
         Filter f
         [,int &accepted,
            int &rejected])
```

**Operation**

Applies the filter in the current module, or to module `m` where it is specified. The third and fourth parameters return the number of objects accepted and rejected respectively under the filter.

stringOf(filter)

**Declaration**

```c
string stringOf(Module m,
                 Filter f)
```

**Operation**

Returns a string representation of filter `f` in module `m`.

ancestors(show/hide)

**Declaration**

```c
void ancestors(bool show)
```

**Operation**

Shows filtered object ancestors if `show` evaluates to `true`. Hides filtered object ancestors if `show` evaluates to `false`.

ancestors(state)

**Declaration**

```c
bool ancestors(Module myModule)
```

**Operation**

Returns `true` if filtered object ancestors are showing in the specified module. Returns `false` if filtered object ancestors are not showing in the specified module.
applyFiltering

Declaration

void applyFiltering(Module)

Operation

Sets the module explorer display to reflect the current filter applied to the specified module.

unApplyFiltering

Declaration

void unApplyFiltering(Module)

Operation

Switches off filtering in the module explorer for the specified module.

applyingFiltering

Declaration

bool applyingFiltering(Module)

Operation

Returns a boolean indicating whether filtering is turned on in the module explorer for the specified module.

Filters example program

// filter DXL example
/*
  example program building DXL filters
  can be used in Car Project "Car user reqts"
  module.
*/

// "show" sets the passed filter, refreshes
// the screen and waits for the next filter.

void show(Filter f, bool last, string what) {
    set f
    refresh current
if (!last) what = what "\n\nready for next filter?"
    ack what
}
load view "Collect reqts"  // if present
filtering on
// declare a filter
Filter f1 = attribute "Acceptability" == "Acceptable"
// display it
show(f1, false, "Acceptability == Acceptable")
// a compound filter
Filter f2 = f1 && attribute "Priority" != "luxury"
show(f2, false, "previous filter && Priority != luxury")
Filter f3 = excludeLeaves
show(f3, true, "exclude Leaves")
filtering off

---

**Compound filters**

Compound filters can be constructed.

**Syntax**

Filter compound = Filter 1 operator Filter 2
                    [operator Filter 3]...

where:

<table>
<thead>
<tr>
<th>compound</th>
<th>is a variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator</td>
<td>is one of:</td>
</tr>
<tr>
<td></td>
<td>&amp;&amp; meaning AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>! meaning NOT</td>
</tr>
<tr>
<td>1 2 3</td>
<td>are strings</td>
</tr>
</tbody>
</table>
Operation
Combines filters to create a complex filter.

Examples
This example filters on those objects that contain the words shall or must, regardless of case.

Filter required = contents("shall", false) || contents("must", false)

This example shows the use of the getCompoundFilterType_ and getComponentFilter_ filter functions. See the sections below this example for the declarations and operations of these functions.

Filter f1 = attribute "Object Heading" == ""
Filter f2 = contents("shall", false) || contents("must", false)
Filter f = f1 && f2

Module m = current

void decomposeFilter(Filter f,string tabStr)
{
    print tabStr "" stringOf(m, f) \\
    tabStr = tabStr "   "
    int ct = getCompoundFilterType_ f
    if (ct == -1)      print ""
    else if (ct == 0)  print tabStr "AND
    else if (ct == 1)  print tabStr "OR 
    else if (ct == 2)  print tabStr "NOT"
    else               print tabStr "???

    int i = 0
    Filter f1 = getComponentFilter_ (f,i++)
    while (!null f1)
    {
        decomposeFilter(f1, tabStr "   ")
        f1 = getComponentFilter_ (f,i++)
    }
}
decomposeFilter(f,""
The output is as follows:

(Object Heading is empty) AND ((Contains 'shall') OR (Contains 'must'))

AND

Object Heading is empty

(Contains 'shall') OR (Contains 'must')

OR

Contains 'shall'

Contains 'must'

In the special case where a user-defined attribute used in a filter has been deleted or an enumeration value used has been deleted, the filter clause will be omitted.

Filter f1 = attribute "theEnum" == "A"
Filter f2 = contents("shall", false) || contents("must", false)
Filter f = f1 && f2

When theEnum type has a value 'A', the following is output:

(theEnum == A) AND ((Contains 'shall') OR (Contains 'must'))

AND

TheEnum == A

(Contains 'shall') OR (Contains 'must')

OR

Contains 'shall'

Contains 'must'

If the filter is saved in a view and then the enumeration 'A' is deleted, when the view is reloaded, the output is as follows:

Filter f = current //i.e. the view - we cannot generate a filter with invalid values.

((Contains 'shall') OR (Contains 'must'))

AND
(Contains 'shall') OR (Contains 'must')

OR

Contains 'shall'
Contains 'must'

where the AND clause appears to have only one input.

compoundFilterType_

**Declaration**

```c
int getCompoundFilterType_(Filter)
```

**Operation**

This perm can be used to decompose compound filters into their component parts for analysis, and potential modification or replacement.

Returns an integer value indicating the type of the specified filter.

It returns one of the following new DXL constant values for compound filter types:

```c
int filterTypeAnd
int filterTypeOr
int filterTypeNot
```

It returns -1 for a simple filter. The test for a negative value suffices to indicate that the filter is not compound, as the new constants are all positive values.

If no filter is supplied, a run-time DXL error is generated.

**Example**

See examples in Compound filters above.

compoundFilter

**Declaration**

```c
Filter getComponentFilter_(Filter, int index)
```

**Operation**

This perm can be used to decompose compound filters into their component parts for analysis, and potential modification or replacement.

Returns an integer value indicating the type of the specified filter.

It returns one of the following new DXL constant values for compound filter types:

```c
int filterTypeAnd
int filterTypeOr
```
int filterTypeNot

This perm returns a component filter that is part of the supplied compound filter. If the compound filter is of type
filterTypeNot, the index must be zero, or the perm returns null. If the compound filter is of type
filterTypeOr or filterTypeAnd, an index of 0 or 1 returns the first or second sub-filter, and any other index
value returns null.

If the supplied filter is not a compound filter, the perm returns null.

If no filter is supplied, a run-time DXL error is generated.

Example
See examples in Compound filters above.

Filtering on multi-valued attributes

This section defines the functions that can be used to filter on multi-valued attributes.

includes

Declaration
Filter includes(attribute(string attrName),
               string s)

Operation
Returns the definition of a simple filter on a multi-valued attribute named attr, where s contains the filtering value.
If the attribute contains s, it is included in the filter set. The string s can only contain one value.

Example
This example filter set includes all objects with multi-valued attributes, one value of which is "ABC":
Filter f1 = includes(attribute "attribute name",
      "ABC")
set f1
filtering on

excludes

Declaration
Filter excludes(attribute(string attrName),
               string s)
Operation
Returns the definition of a simple filter on a multi-valued attribute, where \( s \) contains the filtering value.
If the attribute contains \( s \), it is excluded from the filter set. The string \( s \) can only contain one value.

Example
This example filter set excludes all objects with multi-valued attributes, one value of which is "ABC":
Filter \( f2 = \text{excludes(attribute "attribute name", "ABC")} \)
set f2
filtering on

Sorting modules

This section defines the operators and functions that allow you to sort a formal module in a similar way to the Rational DOORS user interface. These language elements use the data type Sort.

ascending

Declaration
Sort ascending(string \( attrName \))

Operation
Returns a type Sort, which sorts the current display with respect to the values of the object attribute named \( attrName \), in ascending order.

Rational DOORS always refreshes the current module at the end of a script's execution. If a sorted display is to be viewed before that time, you must call refresh current Module.

Example
set ascending "Absolute Number"
sorting on

descending

Declaration
Sort descending(string \( attrName \))

Operation
Returns a type Sort, which sorts the current display with respect to the values of the object attribute named \( attrName \), in descending order.
Rational DOORS always refreshes the current module at the end of a script’s execution. If a sorted display is to be viewed before that time, you must call refresh current Module.

**Example**

set descending "Absolute Number"
sorting on

---

### Compound sort

Compound sort rules can be constructed, as shown in the following syntax:

**Syntax**

```dxl
Sort compound = Sort 1 && Sort 2
```

where:

- `compound` is a variable
- `&&` means AND
- `1 2` are strings

**Operation**

Combine a first sort with a second sort which discriminates between the objects that share the same value in the first sort.

**Example**

This example sorts by the user who created the object and then by the most recently created objects:

```dxl
Sort compound = ascending "Created By" &&
               descending "Absolute Number"
```

set compound
sorting on

---

### set(sort)

**Declaration**

```dxl
void set([Module m,]
         Sort s)
```

**Operation**

Applies the sort rule `s` in the module specified by `m` or, if `m` is omitted, in the current module. The command `sorting` can be used to display sorted output in the current module.
sorting

Declaration
void sorting(bool onOff)

Operation
Displays sorted output in the current module.

Example
set descending "Absolute Number"
sorting on

stringOf(sort)

Declaration
string stringOf(Sort s)

Operation
Returns a string representation of sort s in the current module.

isAscending

Declaration
bool isAscending(Column c)

Operation
Determines whether a column c is sorted in ascending order. If the column is not sorted then false is returned.

isDescending

Declaration
bool isDescending(Column c)

Operation
Determines whether a column c is sorted in descending order. If the column is not sorted then false is returned.
for sort in sort

Declaration

for s in sr do {
  ...
}
where

s is a variable of type Sort
sr is a variable of type Sort

Operation

Assigns \( s \) to be each successive sort in a given compound sort \( sr \).

Example

This example prints all sorting information for the currently defined sort in the current module. Must be run from an open module.

Sort sr = current
Sort s
for s in sr do {
  print stringOf s"\n"
}

destroySort

Declaration

void destroySort(Module m)

Operation

This perm removes any sort criteria stored with the specified module \( m \).

Sorting example program

// sort DXL example
/*
  example program building DXL sorts
*/
Sort s1 = ascending "Absolute Number"
Sort s2 = descending "Absolute Number"
sorting on
refresh current
set s1
ack "hello"
set s2
refresh current
ack "hello"
set s1
refresh current
ack "hello"
set s2
refresh current

Views

This section defines functions and a for loop for building and manipulating Rational DOORS views. Some of these elements use the View data type, which is a handle created for use by other functions.

If a view is to be created you must make sure that the module is in display mode.

The standard view is displayed by default. It cannot be altered or deleted, but can be loaded.

currentView

Declaration
string currentView(Module m)

Operation
Returns the name of the view that is currently selected for the given module.

descendants(show/hide)

Declaration
void descendants(bool expression)

Operation
Shows descendants in the module window if expression is true. Hides descendants if expression is false.
descendants(state)

Declaration
bool descendants(Module m)

Operation
Returns true if the current view in module m is set to show descendants; otherwise returns false.

view

Declaration
View view([Item item,] string viewName)

Operation
Returns a handle to a specific view in item, or if item is omitted, the current module. The item argument must have the value Formal or Descriptive (a formal or descriptive module). If item is any other value, the function returns null.

The view need not exist; if it does not, a new view is created but not saved until the save (view) function is called.

delete(view)

Declaration
string delete([Module m,] View v)

Operation
Deletes the view having handle v from module m, or if m is omitted, from the current module. The returned string is non-NULL on error, else NULL.

Example
View v = view("Basic view")
string s = delete(v)

setPreloadedView

Declaration
bool setPreloadedView(ViewDef view, string name)
Operation
Sets the preloaded view name for the specified ViewDef `view`. Returns `true` on success, and `false` on failure. Will fail and generate a run-time DXL error if there is no current module. Will also fail if the specified name does not designate a view in the current module to which the current user has Read access.

Note that this perm does not check the relative access controls on the inheriting and inherited views, because the ViewDef `view` does not include access controls. These checks are made if and when the ViewDef settings are saved using the change or save perm.

**preloadedView**

Declaration

```plaintext
string preloadedView(ViewDef view)
```

Operation

Returns the preloaded view name for the specified ViewDef `view`. Returns a null (empty) string if no preloaded view is specified for this ViewDef `view`, or if the current user does not have read access to the inherited view. Generates a run-time DXL error and returns an empty string if there is no current module.

**isinheritedView**

Declaration

```plaintext
bool isInheritedView(string viewName)
```

Operation

This returns `true` if any view in the current module is configured to inherit settings from a view whose name matches the supplied string `viewName`, and to which the current user has read access. The user does not need to have read access to the inheriting view for this perm to return `true`. It generates a run-time DXL error and returns `false` if there is no current module.

**isValidName**

See “isValidName,” on page 285.

**linkIndicators(show/hide)**

Declaration

```plaintext
void linkIndicators(bool show)
```

Operation

Shows the link indicators in the current module if `show` evaluates to `true`. Hides the link indicators in the current module if `show` evaluates to `false`. 
**linkIndicators(state)**

**Declaration**

```dxl
bool linkIndicators(Module myModule)
```

**Operation**

Returns `true` if link indicators are showing in the specified module. Returns `false` if link indicators are not showing in the specified module.

---

**load**

**Declaration**

```dxl
bool load([Module m,] View v)

bool load(Module m, View v, bool queryUnsavedChanges)
```

**Operation**

Attempts to load the view handle `v` in module `m`, or if `m` is omitted, in the current module. Supports loading the standard view. If the function fails, it returns `false`.

If the `Module` parameter is supplied, then supplying the `queryUnsavedChanges` flag is also possible. If set to `true`, and the view load will cause unsaved changes in the current view to be lost, and the users settings indicate that they wish to be informed when view changes will be lost, a confirmation query will be given to the user. The view will not be loaded if the user indicates they do not wish to lose the changes. This flag will only have an effect if the module is visible.

**Example**

```dxl
load view "cost analysis"
load view "Standard view"
```

---

**name(view)**

**Declaration**

```dxl
string name(View view)
```

**Operation**

Returns the name of `view`.

---

**next, previous(filtered)**

**Declaration**

```dxl
Object next(Object o,
            Filter filter)
```
Object previous(Object o,  
       Filter filter)

Operation
These functions return the next or previous object at the current level of hierarchy that matches filter.

clearDefaultViewForModule

Declaration
string clearDefaultViewForModule(Module m)

Operation
Clears the default view setting for the specified module. Returns a null string if the operation succeeds; otherwise, returns an error message.

clearDefaultViewForUser

Declaration
string clearDefaultViewForUser(Module m)

Operation
Clears the default view setting, for the current user, for the specified module. Returns a null string if the operation succeeds; otherwise, returns an error message.

getDefaultViewForModule

Declaration
string getDefaultViewForModule(Module m)

Operation
Returns the name of the default view for the specified module. If no default is specified, returns null.

getDefaultViewForUser

Declaration
string getDefaultViewForUser(Module m)

Operation
Returns the name of the default view for the current user, for the specified module. If no default is specified for the current user, returns null.
### save(view)

**Declaration**

```
void save(View v)
```

**Operation**

Saves the view having handle `v` in the current module.

### setDefaultViewForModule

**Declaration**

```
string setDefaultViewForModule(Module m,
    string viewName)
```

**Operation**

Sets the default view for the specified module to `viewName`. Returns a null string if the operation succeeds; otherwise, returns an error message.

### setDefaultViewForUser

**Declaration**

```
string setDefaultViewForUser(Module m,
    string viewName)
```

**Operation**

Sets the default view, for the current user, for the specified module, to `viewName`. Returns a null string if the operation succeeds; otherwise, returns an error message.

### showDeletedObjects(get)

**Declaration**

```
bool showDeletedObjects(void)
```

**Operation**

Returns `true` if the current view in the current module is set to show deleted objects; otherwise returns `false`. 
showDeletedObjects(show/hide)

Declaration

void showDeletedObjects(bool show)

Operation

Shows deleted objects in the module window if show is true. Hides deleted objects if show is false.

showChangeBars(get)

Declaration

bool showChangeBars(Module module)

Operation

Returns true if the specified module shows object change bars. Otherwise, returns false.

showChangeBars(show/hide)

Declaration

void showChangeBars(bool show)

Operation

Sets the option for showing object change bars in the current module.

showGraphicsDatatips(get)

Declaration

bool showGraphicsDatatips(Module module)

Operation

Returns true if the specified module shows datatips in Graphics Mode. Otherwise, returns false.

showGraphicsDatatips(show/hide)

Declaration

void showGraphicsDatatips(bool show)

Operation

Sets the option for showing datatips in Graphics Mode in the current module.
showGraphicsLinks(get)

Declaration
bool showGraphicsLinks(Module module)

Operation
Returns true if the specified module shows links in Graphics Mode. Otherwise, returns false.

showGraphicsLinks(show/hide)

Declaration
void showGraphicsLinks(bool show)

Operation
Sets the option for showing links in Graphics Mode in the current module.

showingExplorer

Declaration
bool showingExplorer(Module module)

Operation
Returns true if the specified module is showing the Module Explorer. Otherwise, returns false.

showExplorer, hideExplorer

Declaration
void showExplorer(Module module)
void hideExplorer(Module module)

Operation
Sets the specified module to show or hide the Module Explorer.

showPrintDialogs(get)

Declaration
bool showPrintDialogs()
**Operation**

Gets the current setting for displaying print dialog boxes.

Printing from the Rational DOORS user interface, rather than from DXL, automatically sets `showPrintDialogs` back to `true`.

### showPrintDialogs(set)

**Declaration**

```plaintext
void showPrintDialogs(bool onOff)
```

**Operation**

Sets whether print dialog boxes should be displayed when printing from DXL. This includes the printer selection dialog box, the warnings issued when printing in graphics view, or in a view that spans more than one page.

When `showPrintDialogs` is turned off, the printer selection dialog box is not displayed, so the default Windows printer, or the printer referred to in the appropriate environment variable on UNIX, is used for all printed output.

### for view in module

**Syntax**

```plaintext
for s in views(Module m) do {
    ...
}
```

where:

- `s` is a string variable
- `m` is a module of type `Module`

**Operation**

Assigns the string `s` to be each successive view name in the module `m`.

**Example**

This example prints all views in the current module:

```plaintext
string name
for name in views current Module do
    print name "\n"
```
canInheritView

Declaration

string canInheritView(View v1, View v2, bool &b)
string canInheritView(ViewDef vd1, View v2, bool &b)

Operation

The first form returns true if view v1 can inherit settings from view v2 in the current module according to access control restrictions. The restrictions are that every user who has read access to v1 must also have read access to v2.
The second form determines whether a ViewDef can inherit settings from a View.
In both cases an error is returned on failure, or null on success.
Note that the test does not take into account group membership, so a user who is given specific access to v1 or vd1 and who is granted access to v2 by virtue of group membership will not qualify v2 as inheritable.

clearInvalidInheritanceOf

Declaration

bool clearInvalidInheritanceOf(string viewname[, ViewDef vd])

Operation

This clears the preloadView setting of any views which currently inherit settings from the named view in the current module if that inheritance is invalid according to the access rights constraints as reflected by the canInheritView perm. It returns true on success and false on failure, and generates a run-time DXL error if there is no current module or if the views index file cannot be locked.
If the defn argument is specified, then the validity test is applied as if the named view had the access controls in the defn argument.

invalidInheritedView

Declaration

bool invalidInheritedView(string viewname[, ViewDef vd])

Operation

This returns true if any view in the current module is configured to inherit settings from a view of the specified name to which the current user has read access, and the access control restrictions applied by canInheritView prohibit the inheritance relationship. The user does not have to have read access to the inheriting view. If a ViewDef is specified, then the restrictions are those which would apply if the view had the access rights contained within it.
**setViewDescription**

**Declaration**

void setViewDescription(ViewDef vd, string desc)

**Operation**

Sets the description for a view where `vd` is the view definition handle.

**getViewDescription**

**Declaration**

string getViewDescription(ViewDef vd)

**Operation**

Returns the description for a view where `vd` is the view definition handle.

**for View in View**

**Syntax**

for View1 in View2 do {
    ...
}

**where:**

- **View1** is a variable of type View
- **View2** is a variable of type View

**Operation**

Assigns `View1` to be each successive View whose settings can be inherited by the specified `View2` according to the same access control restrictions applied by the `canInheritView` perm.
View access controls

canCreate(view)

Declaration

bool canCreate(ModName_ modRef, View v)

Operation

Returns true if the current Rational DOORS user has create access to view v, which can be specified as in the module modRef. Otherwise, returns false.

canControl(view)

Declaration

bool canControl(ModName_ modRef, View v)

Operation

Returns true if the current Rational DOORS user can change the access controls on view v, which is specified as module modRef. Otherwise, returns false.

canRead(view)

Declaration

bool canRead(ModName_ modRef, View v)

Operation

Returns true if the current Rational DOORS user can read view v, which is specified as the module modRef. Otherwise, returns false.

canModify(view)

Declaration

bool canModify(ModName_ modRef, View v)

Operation

Returns true if the current Rational DOORS user can modify view v, which is specified as module modRef. Otherwise, returns false.
canDelete(view)

Declaration

bool canDelete(ModName_ modRef, View v)

Operation

Returns true if the current Rational DOORS user can delete view v, which is specified as module modRef. Otherwise, returns false.

canWrite(view)

Declaration

bool canWrite(ModName_ modRef, View v)

Operation

Returns true if the current Rational DOORS user can write view v, which is specified as the module modRef. Otherwise, returns false.

Views example program

// view DXL example
/* construct a new view containing a selection of attributes. Save as the view "View DXL example". */
string viewName = "View DXL example"
DBE attrList
// contains selection of attributes to display
void buildFn(DBE dbe) {
// construct view of attributes chosen
    string attr
    Column c
    int n = 0 // number of existing columns
    int i    // column index
    for c in current Module do n++ // count the columns
    for i in 1:n do
        delete(column 0) // delete n column 0s
View definitions

This section defines functions that use the ViewDef data type, which holds all the settings from the Advanced tab of the Views dialog box, such as compression and outlining.
**create(view definition)**

**Declaration**

```c
ViewDef create([Module m,
    bool allSettings])
```

**Operation**

Creates a view definition in the module `m`, or if no arguments are supplied, in the current module. The `allSettings` argument specifies whether by default all of the view settings are saved.

**createPrivate**

**Declaration**

```c
ViewDef createPrivate([Module m, bool allSettings])
```

**Operation**

This new perm creates a new private module view. It saves the view with non-inherited access, giving the current user full access and everyone else no access.

**createPublic**

**Declaration**

```c
ViewDef createPublic([Module m, bool allSettings])
```

**Operation**

This new perm creates a new public module view. It saves the view with non-inherited access, giving the current user full access and everyone else Read access.

**get(view definition)**

**Declaration**

```c
ViewDef get([Module m,]
    View v)
```

**Operation**

Returns the underlying view definition in `v` for the specified module, or if `m` is omitted, for the current module.
change(view definition)

Declaration
ViewDef change(View v,
    ViewDef viewDef
    [,string viewName])

Operation
Changes the underlying view definition in v. Optionally, changes the name of the view.

delete(view definition)

Declaration
void delete(ViewDef viewDef)

Operation
Deletes the view definition viewDef from the current module. The returned string is non-NULL on error, else NULL.

Example
View v = view("Basic View")
ViewDef vdef = get(v)
string s = delete(current Module, v)

save(view definition)

Declaration
void save([Module m,]
    View v,
    ViewDef viewDef)

Operation
Saves the view definition viewDef into view v in module m, or if m is omitted, in the current module.

useAncestors(get and set)

Declaration
bool useAncestors(ViewDef viewDef)
void useAncestors(ViewDef viewDef,
    bool save)
Operation
The first form returns true if the option to save the advanced filter option for showing ancestors is currently set. Otherwise, returns false.

The second form sets the option to save the advanced filter option for showing ancestors.

If the option for showing ancestors is set, a filtered view contains objects that match the given filter and that object’s parent hierarchy too.

useDescendants(get and set)

Declaration
bool useDescendants(ViewDef viewDef)
void useDescendants(ViewDef viewDef,
                      bool save)

Operation
The first form returns true if the option to save the advanced filter option for showing descendants is currently set. Otherwise, returns false.

The second form sets the option to save the advanced filter option for showing descendants.

If the option to show descendants is set, a filtered view contains objects that match the given filter and that object’s child hierarchy too.

useCurrent(get and set)

Declaration
bool useCurrent(ViewDef viewDef)
void useCurrent(ViewDef viewDef,
                bool save)

Operation
The first form returns true if the option to save information about the currently selected object is currently set. Otherwise, returns false.

The second form sets the option to save information about the currently selected object.

useSelection(get and set)

Declaration
bool useSelection(ViewDef viewDef)
void useSelection(ViewDef viewDef,
                  bool save)
Operation

The first form returns true if the option to save information about currently selected objects is currently set. Otherwise, returns false.

The second form sets the option to save information about currently selected objects.

useColumns(get and set)

Declaration

bool useColumns(ViewDef viewDef)
void useColumns(ViewDef viewDef,
       bool save)

Operation

The first form returns true if the option to save column information is currently set. Otherwise, returns false.

The second form sets the option to save column information.

useFilterTables(get and set)

Declaration

bool useFilterTables(ViewDef viewDef)
void useFilterTables(ViewDef viewDef,
       bool save)

Operation

The first form returns true if the option to save the advanced filter option for hiding non-matching table cells is currently set. Otherwise, returns false.

The second form sets the option to save the advanced filter option for hiding non-matching table cells.

useGraphicsColumn(get and set)

Declaration

bool useGraphicsColumn(ViewDef viewDef)
void useGraphicsColumn(ViewDef viewDef,
       bool save)

Operation

The first form returns true if the option to save information about which column’s values are displayed in the object boxes when in Graphics Mode is currently set. Otherwise, returns false.

The second form sets the option to save information about which column’s values are displayed in the object boxes when in Graphics Mode.
useShowExplorer(get and set)

**Declaration**

```c
bool useShowExplorer(ViewDef viewDef)
void useShowExplorer(ViewDef viewDef,
                    bool save)
```

**Operation**

The first form returns `true` if the option to save the **Module Explorer** setting (shown on the **View** menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the **Module Explorer** setting.

useGraphics(get and set)

**Declaration**

```c
bool useGraphics(ViewDef viewDef)
void useGraphics(ViewDef viewDef,
                 bool save)
```

**Operation**

The first form returns `true` if the option to save the **Graphics Mode** setting (shown on the **View** menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the **Graphics Mode** setting.

useOutlining(get and set)

**Declaration**

```c
bool useOutlining(ViewDef viewDef)
void useOutlining(ViewDef viewDef,
                  bool save)
```

**Operation**

The first form returns `true` if the option to save the **Outline** setting (shown on the **View** menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the **Outline** setting.
useCompression(get and set)

**Declaration**

```c
bool useCompression(ViewDef viewDef)
void useCompression(ViewDef viewDef,
                   bool save)
```

**Operation**

The first form returns `true` if the option to save the **Compress** setting (shown on the **View** menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the **Compress** setting.

useLevel(get and set)

**Declaration**

```c
bool useLevel(ViewDef viewDef)
void useLevel(ViewDef viewDef,
             bool save)
```

**Operation**

The first form returns `true` if the option to save the **Level** setting (shown on the **View** menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the **Level** setting.

useSorting(get and set)

**Declaration**

```c
bool useSorting(ViewDef viewDef)
void useSorting(ViewDef viewDef,
                bool save)
```

**Operation**

The first form returns `true` if the option to save the **Sort** setting (shown on the **View > Show** menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the **Sort** setting.
useFiltering(get and set)

**Declaration**

```c
bool useFiltering(ViewDef viewDef)
void useFiltering(ViewDef viewDef,
                  bool save)
```

**Operation**

The first form returns `true` if the option to save the **Filter** setting (shown on the **View > Show** menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the **Filter** setting.

useShowDeleted(get and set)

**Declaration**

```c
bool useShowDeleted(ViewDef viewDef)
void useShowDeleted(ViewDef viewDef,
                    bool save)
```

**Operation**

The first form returns `true` if the option to save the **Deletions** setting (shown on the **View > Show** menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the **Deletions** setting.

useShowPictures(get and set)

**Declaration**

```c
bool useShowPictures(ViewDef viewDef)
void useShowPictures(ViewDef viewDef,
                     bool save)
```

**Operation**

The first form returns `true` if the option to save the **Pictures** setting (shown on the **View > Show** menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the **Pictures** setting.
useShowTables(get and set)

Declaration

```cpp
bool useShowTables(ViewDef viewDef)
void useShowTables(ViewDef viewDef,  
                   bool save)
```

Operation

The first form returns `true` if the option to save the Table Cells setting (shown on the View > Show menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the Table Cells setting.

useShowLinkIndicators(get and set)

Declaration

```cpp
bool useShowLinkIndicators(ViewDef viewDef)
void useShowLinkIndicators(ViewDef viewDef,  
                           bool save)
```

Operation

The first form returns `true` if the option to save the Link Arrows setting (shown on the View > Show menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the Link Arrows setting.

useShowLinks(get and set)

Declaration

```cpp
bool useShowLinks(ViewDef viewDef)
void useShowLinks(ViewDef viewDef,  
                  bool save)
```

Operation

The first form returns `true` if the option to save the Graphics Links setting (shown on the View > Show menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the Graphics Links setting.
useTooltipColumn (get and set)

**Declaration**

```dxl
definition useTooltipColumn(ViewDef viewDef)
void useTooltipColumn(ViewDef viewDef,
                      bool save)
```

**Operation**

The first form returns `true` if the option to save the *Graphics Datatips* setting (shown on the `View > Show` menu) is currently set. Otherwise, returns `false`.

The second form sets the option to save the *Graphics Datatips* setting.

useWindows (get and set)

**Declaration**

```dxl
definition useWindows(ViewDef viewDef)
void useWindows(ViewDef viewDef,
                bool save)
```

**Operation**

The first form returns `true` if the option to save the current window size and position is currently set. Otherwise, returns `false`.

The second form sets the option to save the current window size and position.

If a view is saved in batch mode with the option to save the current window size and position set to `true`, the view will be saved with a module window that is not visible.

useAutoIndentation

**Declaration**

```dxl
void useAutoIndentation(ViewDef vDef, Bool)
definition useAutoIndentation(ViewDef vDef)
```

**Operation**

The first form sets the auto-indentation status of the supplied ViewDef (this equates to the setting of the “Indentation of main column” check box on the advanced tab of the “Manage Views” dialog).

The second form returns the auto-indentation status of the supplied ViewDef.

**Example**

```dxl
ViewDef viewInfo = get(view "viewName")
```
Columns

This section defines functions and a for loop for building and manipulating Rational DOORS columns. These elements use the data types Column and Justification.

**Note:** The data type Justification is used for constants specifying text alignment.

column

**Declaration**

```dxi
Column column([Module m,] int n)
```

**Operation**

Returns a handle on the \( n \)th column, starting from 0, in module \( m \), or if \( m \) is omitted, in the current module. The handle is used in other column functions.

Column alignment constants

The following constants of type Justification are defined for reading or setting the alignment of text in a column.

- **left**
  - Aligns text to the left column
- **right**
  - Aligns text to the right column
- **center**
  - Centers text
- **centre**
  - Centers text
- **full**
  - Justifies text

attribute(in column)

**Declaration**

```dxi
void attribute(Column c, string attr)
```

**Operation**

Makes column \( c \) display the attribute \( attr \).
attrName

Declaration
string attrName(Column c)

Operation
Returns the name of the attribute displayed in a column; this is the value that the attribute(in column) function sets. Returns null if the column does not display an attribute.

Example
This example prints out the names of all the attributes displayed in the current view:

Module m = current
Column c
for c in m do {
    print "<" (attrName c) "">
}

color(get)

Declaration
string color(Column c)

Operation
Returns the name of the attribute used for coloring a column, or null if none is set.

color(set)

Declaration
void color(Column c,
        string attrName)

Operation
Uses color on column c as specified by the attribute named attrName.

backgroundColor(get)

Declaration
string backgroundColor(Column c)
Operation

Returns the name of the attribute that is used to color the background in column c, or null if none is set.

backgroundColor(set)

Declaration
void backgroundColor(Column c, string enumAttrName)

Operation
Sets the attribute enumAttrName as the background color for column c.

Example
Column c = column(1)
backgroundColor(c, "enumAttrName")
Sets the background color for column c to use attribute enumAttrName.

Column c = column(1)
backgroundColor(c, "")
Removes any previously configured background color attribute for column c.

delete(column)

Declaration
void delete(Column c)

Operation
Deletes column c. This command should not be used inside the column for..do loop. If every column must be deleted, use the following example.

Example
int n = 0       // number of existing columns
int i           // column index
Column c
for c in current Module do
    n++  // count the columns
// delete n column 0s
for i in 1:n do {
    delete column 0
}
dxl(get)

Declaration

string dxl(Column c)

Operation

Returns the DXL code set for DXL column c.

Example

Column col

for col in current Module do {
    string att = attrName(col)
    if (null att) {
        if (main(col)) {
            print "main\t"
        } else {
            print dxl(col) "\t"
        }
    } else {
        print att "\t"
    }
}

dxl(set)

Declaration

void dxl(Column c, string dxlCode)

Operation

Sets the DXL code to use in a DXL column. This is equivalent to the menu option Column > Edit > dxl.

If you wish to use a Windows-style file separator (\), you must duplicate it (\\) so that DXL does not interpret it as a meta character in the string. Because Rational DOORS automatically converts UNIX-style file separators (/) for Windows, it is usually more convenient to use them.

Example

dxl(column 0, "display obj."Object Heading")
dxl(column 1, "#include <layout/trace.dxl>")
graphics(get)

Declaration
bool graphics(Column c)

Operation
Returns true when c is the column nominated for viewing in a graphics display; otherwise, returns false.

graphics(set)

Declaration
void graphics(Column c)

Operation
Nominates the column for viewing in a graphics display.

info(get)

Declaration
bool info(Column c)

Operation
Returns true when c is the column nominated for use by the datatips mechanism in Graphics mode; otherwise, returns false.

info(set)

Declaration
void info(Column c)

Operation
Nominates the column for use by the datatips mechanism in Graphics mode.

insert(column in module)

Declaration
Column insert(Column c)
Operation
Inserts a column, pushing subsequent columns one right. Returns a handle to the new column. If a column is inserted at a new position, it is important to initialize the width of the new column (see the width(get) function).

Example
This example inserts a new column 1 as a copy of the old column 1, if present:

insert(column 1)

justify(get alignment)

Declaration
string justify(Column c)
Justification justify(Column c)

Operation
The first form returns a string version of the type Justification constants.
The second form returns the type Justification constant for the specified column c. The constants are defined in “Column alignment constants,” on page 688.

justify(set alignment)

Declaration
void justify(Column c, Justification j)

Operation
Sets the alignment or justification of column c to the Justification constant j, which can be one of the constants defined in “Column alignment constants,” on page 688.

Example
justify(column 1, center)

main(get)

Declaration
bool main(Column c)

Operation
Returns true if the column is the main text column (with the appearance of the second column in the standard view).
main(set)

Declaration
void main(Column c)

Operation
Makes column c the main text column (with the appearance of the second column in the standard view).

link(get)

Declaration
bool link(Column c)

Operation
Returns true if column c is a link indicator column.

link(set)

Declaration
void link(Column c)

Operation
Makes column c a link indicator column.

changebar(get)

Declaration
bool changebar(Column c)

Operation
Returns true if column c is a change bar column.

changebar(set)

Declaration
void changebar(Column c)

Operation
Makes column c a change bar column.
**text(column)**

**Declaration**

```dxl
string text(Column c, 
          Object o)
```

**Operation**

Returns the text contained in column `c` for object `o`.

When `c` is the main column, this function returns the empty string. You must assemble the elements of the main column from the "Object Heading" and "Object Text" attributes, and the number function. You can use the `main(get)` function to check for this condition.

**Example**

```dxl
Object o
Column c

for o in current Module do {
    for c in current Module do {
        if (main c) {
            print o."Object Heading" "\n\n"
            print o."Object Text" "\n\n"
        } else {
            print text(c, o) "\n"
        }
    }
}
```

**title(get)**

**Declaration**

```dxl
string title(Column c)
```

**Operation**

Returns the string that is the title of column `c`.

**title(set)**

**Declaration**

```dxl
void title(Column c, 
          string heading)
```

**Operation**

Sets the title of column `c` to the string `heading`. 
width(get)

Declaration

int width(Column c)

Operation

Returns the number of screen pixels used by column c.

Example

Column c
for c in current Module do {
    print (title c) " " (justify c) " "
    print (width c) "\n"
}

width(set)

Declaration

void width(Column c, int w)

Operation

Sets the width of column c to w in pixels.

currentColumn(get)

Declaration

Column currentColumn(void)

Operation

Gets the current column for this DXL context. If the DXL is not layout DXL then this will return NULL.

Example

string name = ?attributeName?;
Column c = currentColumn()

if (null c)
{
    // Attribute DXL
obj.attrDXLName = name;
}
else
{
    // Layout DXL
    display(name);
}

---

for columns in module

Syntax
for c in m do {
    ...
}
where:

- \( c \) is a variable of type Column
- \( m \) is a module of type Module

Operation
Assigns the variable \( c \) to be each successive column in the current view in module \( m \). The command delete(Column) should not be used inside the body of the loop.

Example
Column c
for c in current Module do print (title c) "\n"

---

Scrolling functions

This section gives the syntax for scrolling functions, which control view scrolling. The functions use internal data types, so declarations are not stated.

scroll

Syntax
scroll(position)
Operation

scrolls to the position determined by the supporting functions: top, bottom, to, up, down, and page. The syntax for these functions is as follows:

top(Module m)
bottom(Module m)
to({top|bottom}(Module m))
to(Object o)
up(Module m)
down(Module m)
page({up|down}(Module m))

Example

scroll up current Module
scroll down current Module
scroll page up current Module
scroll page down current Module

Layout DXL

This section describes the DXL features unique to layout DXL. Layout DXL is used to populate a column within a Rational DOORS view, typically to construct traceability or impact analysis reports.

The Insert Column dialog box in the Rational DOORS user interface has a Layout DXL option which pops up a DXL Library window, which enables you to browse several layout DXL programs; they can also be found in $DOORSHOME/lib/dxl/layout.

For information on how to check the validity of your DXL code, see the checkDXL function.

Layout context

Layout DXL programs run in a context where the variable obj is pre-declared. You can have a column that contains DXL code. The code calculates the value to display for each object. The current object to calculate is referred to as obj.

display

Declaration

void display(string line)
void display(attrRef)
where \textit{attrRef} is in one of the following formats:

\begin{itemize}
  \item \texttt{(Object \ o).\(\text{string \ attrName}\)}
  \item \texttt{(Module \ m).\(\text{string \ attrName}\)}
  \item \texttt{(Link \ l).\(\text{string \ attrName}\)}
\end{itemize}

\textbf{Operation}

Adds a new line to the column. The new line contains the specified string or the value of the referenced attribute.

If you pass a referenced attribute and the value contains rich text markup, this function interprets the rich text markup.

If you pass a string that contains rich text markup, this function does not interpret the rich text markup; it passes the rich text tags as text characters. Use the \texttt{displayRich} function if you want the rich text markup to be interpreted.

For more information, see “Rich text,” on page 833.

\textbf{Example}

This example takes two attribute values from the current object \texttt{obj} and calculates a derived value for display. The empty string is needed to convert the area's \texttt{int} value into a string:

```dxl
// calculate area of \texttt{obj}
int length = obj."Length"
int width = obj."Width"
display \((\text{length*width})\ "\"
```

This example adds another line to the column with the attribute "Object Text":

```dxl
display \texttt{obj."Object \ Text"}
```

This example is for a module where column A is text and column B is the following DXL:

```dxl
string \texttt{s=obj."A" "\"
display "<"\texttt{s} \texttt{s}>"
```

For values of A of \texttt{hello} and \texttt{bye}, you see:

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{Column: ID} & \textbf{A (text) B (DXL)} \\
\hline
\textbf{Values:} & 1 hello <hellohello> \\
& 2 bye <byebye> \\
\hline
\end{tabular}
\end{center}

\textbf{displayRich}

\textbf{Declaration}

\begin{verbatim}
void displayRich(string richTextString)
\end{verbatim}

\textbf{Operation}

Adds a new line containing the specified string to the column followed by a trailing blank line.

This function operates in the same way as the \texttt{display} function, except that it interprets any rich text markup in the specified string. For more information, see “Rich text,” on page 833.
Example
displayRich richText obj."Object Text"

displayRichWithColo[u]r

Declaration
void displayRichWithColo[u]r(string richTextString)

Operation
Like the existing displayRich for displaying text in layout DXL, but respects the text color specified in the string argument, which must be RTF (not plain text).

Example
bool fullRTF=true
Buffer b1=create
Buffer b2=create
Buffer res=create

b1=obj."Specification"
b2=obj."Proposed_specification"

diff(res, b1, b2, fullRTF)
displayRichWithColour stringOf(res)

getCanvas

Declaration
DBE getCanvas()

Operation
Returns a handle to a virtual canvas with which normal canvas drawing functions can be used. The canvas is in the Rational DOORS formal module display in a column driven by layout DXL. When using this perm, checks should be made for the perm returning a null value, to prevent DXL errors.

hasPicture/exportPicture

Declaration
bool hasPicture(Column c)
string exportPicture(Column c, Object o, string filename, int format)

Operation

The two perms here are for use along with the htmlText perm. After calling htmlText on a column, hasPicture will tell you whether a picture of some layout DXL has been stored with the column. You can then call exportPicture to export the picture.

isFirstObjectInDXLSet(Object)

Declaration

bool isFirstObjectInDXLSet(Object o)

Operation

This perm has been added for use only within layout DXL.

Exposes to DXL the processing of objects during various stages of the execution of layout DXL.

Layout DXL processes sets of objects at a time. When layout DXL is running against an object, that object might be in a set of objects that will be processed. For example, during the repaint of a formal module display, the set is those objects that will be drawn to the display.

This perm returns true in the following cases:

• Layout DXL is not executing
• Layout DXL is executing against a set of objects, and Object is the first to be processed in that set

An object may be simultaneously the first such object in a set and also the last such object in a set; the set might contain a single object.

This perm, and its partner, isLastObjectInDXLSet(), allow layout DXL to perform certain actions only at the start or end of a particular set processing. This can support the DXL programmer to write more efficient layout DXL.

isLastObjectInDXLSet(Object)

Declaration

bool isLastObjectInDXLSet(Object o)

Operation

This perm has been added for use only within layout DXL.

Exposes to DXL the processing of objects during various stages of the execution of layout DXL.

Layout DXL processes sets of objects at a time. When layout DXL is running against an object, that object might be in a set of objects that will be processed. For example, during the repaint of a formal module display, the set is those objects that will be drawn to the display.

This perm returns true in the following cases:

• layout DXL is not executing
• layout DXL is executing against a set of objects, and Object is the last to be processed in that set
An object may be simultaneously the first such object in a set and also the last such object in a set; the set might be singleton.

This perm, and its partner, isFirstObjectInDXLSet(), allow layout DXL to perform certain actions only at the start or end of a particular set processing. This can support the DXL programmer to write more efficient layout DXL.

Example

Insert a Layout DXL column containing the following:

```dxl
if (isFirstObjectInDXLSet(obj))
{
    display "This is the first object in the module window."
}
else if (isLastObjectInDXLSet(obj))
{
    display "This is the last object in the module window."
}
else
{
    // do nothing
}
```

You can then click your mouse on the bottom right hand corner of the module window and resize to see the perms in operation.

---

## setRefreshDelta

**Declaration**

```dxl
void setRefreshDelta(Column c, int delta)
```

**Operation**

Identifies the column in the view that contains the Layout DXL function. This function also specifies the refresh interval in seconds for the displayed values.

**Example**

In this example, the Layout DXL column is the third column in the view. The column numbering begins with 0, so the value of the third column is 2.

```dxl
Column c
c = column(2) //third column in view
setRefreshDelta(c, 100)
```
getRefreshDelta

_Declaration_

\[\text{in getRefreshDelta(Column } \text{c)}\]

_Operation_

Identifies the column in the view that contains the Layout DXL function. This function returns the refresh interval in seconds for the displayed values.

_Example_

In this example, the Layout DXL column is the third column in the view. The column numbering begins with 0, so the value of the third column is 2.

Column c
\[
c = \text{column(2) //third column in view}
\]

int refreshrate

\[
\text{refreshrate = getRefreshDelta(c)}
\]

print refreshrate "\n"

setManualRefresh

_Declaration_

void setManualRefresh(Column c, bool manualRefresh)

_Operation_

If manualRefresh is set to true, then the layout DXL column will only refresh on F5 and at most every n seconds depending on the refresh interval (see setRefreshDelta). Editing the layout DXL properties will still cause a refresh. If manualRefresh is set to false, then user actions such as scroll or select object will cause a refresh.

isManualRefresh

_Declaration_

bool isManualRefresh(Column c)

_Operation_

Returns true if a manual refresh is enabled for that layout DXL column.
Chapter 26
Partitions

This chapter provides information on Rational DOORS partitions.
- Partition concepts
- Partition definition management
- Partition definition contents
- Partition management
- Partition information
- Partition access

Partition concepts

Any partition operation can be performed through DXL. These operations fall into the following categories:
- Management of partition definitions
- Management of partitions
  - Exporting a partition from the home database
  - Accepting a partition in the away database
  - Adding data to a partition in the away database
  - Returning a partition from the away database
  - Rejoining a partition to the home database

Partition definition management

A partition definition describes the information that is to be included in partition. This is a list of modules, called partition modules, and, for each partition module, a list of attributes, views, and (for link modules) linksets to be included. A partition module is really just a placeholder for the real module, but it is associated with a real module. This manual refers to it as if it were the regular module.

In a partition definition, a set of maximum access rights is associated with each partition module, partition attribute, and partition view. These maximum access rights determine what users at the away database can do when the partition definition is used to create a partition.
create(partition definition)

Declaration

PartitionDefinition create(Project p,  
  string name,  
  string desc)

Operation

Creates a partition definition in project $p$ with name $name$ and description $desc$. The partition definition created must be saved before use.

delete(partition definition)

Declaration

string delete(PartitionDefinition pd)

Operation

Removes the partition definition $pd$ from its project.

If successful, returns a null string; otherwise returns a string containing an error message.

dispose(partition definition)

Declaration

string dispose(PartitionDefinition pd)

Operation

Frees up the memory used by DXL to store the partition definition $pd$. It does not affect the partition definition as stored in the database.

If successful, returns a null string; otherwise returns a string containing an error message.

copy(partition definition)

Declaration

string copy(PartitionDefinition pd,  
  string name,  
  string desc)

Operation

Creates a copy of a partition definition with the name $name$, and the description $desc$.

If successful, returns a null string; otherwise returns a string containing an error message.
rename(partition definition)

Declaration

string rename(PartitionDefinition pd,
               string newName)

Operation

Changes the name of a partition definition to newName.
If successful, returns a null string; otherwise returns a string containing an error message.

load(partition definition)

Declaration

PartitionDefinition load(Project p,
                         string name)

Operation

Loads partition definition name in project p. This is used to obtain a handle for editing with the addModule,
addLinkModule, and removeModule functions, but not the addAwayModule, addAwayLinkModule functions.

loadInPartitionDef

Declaration

PartitionDefinition loadInPartitionDef(Project p, string name)

Operation

Loads partition definition associated with the partition name, which is a partition that has been accepted into project p.
This is used in the away database to add data to a partition with the addAwayModule, addAwayLinkModule functions.

save(partition definition)

Declaration

string save(PartitionDefinition pd)

Operation

Saves a partition definition in the home database.
If successful, returns a null string; otherwise returns a string containing an error message.
saveModified(partition definition)

Declaration

```c
string saveModified(Project p,
                    string inPartname,
                    PartitionDefinition pd)
```

Operation

Saves a partition definition in the away database. The partition definition is associated with the partition `inPartname`, which has been accepted into the away database.

If successful, returns a null string; otherwise returns a string containing an error message.

Example

```c
pd = loadInPartitionDef(project, "N")
(...)
saveModified(project, "N", pd)
```

setDescription(partition definition)

Declaration

```c
string setDescription(PartitionDefinition pd,
                      string newDesc)
```

Operation

Changes the description of a partition definition to `newDesc`.

If successful, returns a null string; otherwise returns a string containing an error message.

Partition definition contents

This section describes functions and for loops concerned with the contents of a partition definition.

addModule, addLinkModule

Declaration

```c
string add[Link]Module(PartitionDefinition pd,
                        string modName)
```
Operation

Adds module modName to the partition definition pd. The module name must be specified with a full path name relative to the project (beginning with the project name).

If successful, returns a null string; otherwise returns a string containing an error message.

Use the function addModule for formal modules; use addLinkModule for link modules.

These perms will add the module to the partition definition with access rights set to RMCD by default.

addAwayModule, addAwayLinkModule

Declaration

string addAway[Link]Module(PartitionDefinition pd,
                               string modName)

Operation

Adds module modName to the partition definition pd in the away database. This means that pd must be obtained from the loadInPartitionDef function. The module name must be specified relative to the folder in the away database created when the partition was accepted.

If successful, returns a null string; otherwise returns a string containing an error message.

Use the function addAwayModule for formal modules; use addAwayLinkModule for link modules.

This marks the module as being partitioned in. When the partition is finally returned, the module is returned with the other partitioned-in data.

Example

If you accept a partition called N into a folder B, a folder called N is created inside B. If you then create a module A in folder N, you can add it to the partition definition with:

pd = loadInPartitionDef(project, "N")
addAwayModule(pd, "A")

findModule

Declaration

PartitionModule findModule(PartitionDefinition pd,
                            string modName)

Operation

Returns a handle to the description of the module in the partition definition pd. In the home database, the modName argument must be an absolute path from the containing project (not including the project name). In the away database, the modName argument must be a path relative to the partition folder.
The handle is used with the `findLinkset`, `addLinkset`, `addAwayLinkset`, and `addView`, `addAwayView` functions to edit the information, including linksets, associated with this module in the partition definition.

### findLinkset

**Declaration**

\[
\text{PartitionLinkset findLinkset(PartitionModule } pm, \\
\text{ string } source, \\
\text{ string } target)\
\]

**Operation**

Returns a handle for the linkset between `source` and `target` in the partitioned link module `pm`. The names specified for both the source and target modules must be absolute paths from the containing project (not including the project name).

### findAttribute

**Declaration**

\[
\text{PartitionAttribute findAttribute(PartitionModule } pm, \\
\text{ string } attrName)\
\]

**Operation**

Returns a handle for the attribute called `attrName` in the partition module `pm`. The handle can be used with dot notation to extract the name of the attribute.

### findView

**Declaration**

\[
\text{PartitionView findView(PartitionModule } pm, \\
\text{ string } viewName)\
\]

**Operation**

Returns a handle for the view called `viewName` in the partition module `pm`. The handle can be used with dot notation to extract the name of the view.

### addAttribute, addAwayAttribute

**Declaration**

\[
\text{string add[Away]Attribute(PartitionModule } pm, \\
\text{ string } attrName)\
\]
Operation

Specifies that attribute \textit{attrName} is to be included with the information in partition module \textit{pm}. Use the function \texttt{addAwayAttribute} when adding information in the away database.

If successful, returns a null string; otherwise returns a string containing an error message.

\textbf{addLinkset, addAwayLinkset}

\textbf{Declaration}

\begin{verbatim}
string add[Away]Linkset(PartitionModule pm,
                        string srcName,
                        string trgName)
\end{verbatim}

\textbf{Operation}

Adds a linkset to a partition definition containing \textit{pm}, which must be a link module in the partition definition. The linkset has source \textit{srcName} and target \textit{trgName} in module \textit{pm}.

Use the \texttt{addAwayLinkset} function when adding information in the away database.

For \texttt{addLinkset}, the names specified for both the source and target modules must be absolute paths from the containing project (not including the project name).

For \texttt{addAwayLinkset}, the module name must be specified relative to the folder in the away database created when the partition was accepted.

If successful, returns a null string; otherwise returns a string containing an error message.

\textbf{addView, addAwayView}

\textbf{Declaration}

\begin{verbatim}
string add[Away]View(PartitionModule pm,
                     string viewName)
\end{verbatim}

\textbf{Operation}

Specifies that view \textit{viewName} is to be included with the information in partition module \textit{pm}, which must describe a formal module. Use the function \texttt{addAwayView} when adding information in the away database.

If successful, returns a null string; otherwise returns a string containing an error message.

\textbf{removeModule}

\textbf{Declaration}

\begin{verbatim}
string removeModule(PartitionDefinition pd,
                    string modName)
\end{verbatim}
Operation

Removes a partition module from a partition definition.
If successful, returns a null string; otherwise returns a string containing an error message.

removeAttribute

Declaration

string removeAttribute(PartitionModule pm,
                       string attrName)

Operation

Removes attribute attrName from the information to be included with partition module pm. You cannot remove information from a partition definition in the away database.
If successful, returns a null string; otherwise returns a string containing an error message.

removeLinkset

Declaration

 PartitionLinkset
removeLinkset(PartitionModule pm,
              string source,
              string target)

Operation

Removes a particular linkset from the information to be included with partition module pm, which must be a link module. The names specified for both the source and target modules must be absolute paths from the containing project (not including the project name).

removeView

Declaration

string removeView(PartitionModule pm,
                   string viewName)

Operation

Removes view viewName from the information to be included with partition module pm. You cannot remove information from a partition definition in the away database.
If successful, returns a null string; otherwise returns a string containing an error message.
allowsAccess

Declaration

bool allowsAccess({PartitionAttribute pa| PartitionModule pm| PartitionView pv},
                  PartitionPermission pp)

Operation

Returns true if the data is to be included in the partition with the maximum access rights pp. Otherwise, returns false.

setAccess

Declaration

void setAccess({PartitionAttribute pa| PartitionModule pm| PartitionView pv},
                PartitionPermission pp)

Operation

Sets the maximum access rights to the data in the away database to be pp.

for partition module in partition definition

Syntax

for partModule in partDefinition do {
   ...
}

where:

   partModule is a variable of type PartitionModule
   partDefinition is a variable of type PartitionDefinition

Operation

Assigns partModule to be each successive module within partDefinition.
for partition attribute in partition module

Syntax

for partAttr in partModule do {
    ...
}

where:

partAttr is a variable of type PartitionAttribute
partModule is a variable of type PartitionModule

Operation

Assigns partAttr to be each successive attribute within partModule.

for partition view in partition module

Syntax

for partView in partModule do {
    ...
}

where:

partView is a variable of type PartitionView
partModule is a variable of type PartitionModule

Operation

Assigns partView to be each successive view within partModule.

Partition management

This section describes the functions for exporting, accepting, returning, and rejoining partitions.
apply(partition definition)

Declaration

```dxl
string apply(Project p,
               string partDefName,
               string partName,
               string partDesc,
               string filename[, bool overwrite])
```

Operation

Applies partition definition `partDefName` to create a partition with name `partName` and description `partDesc`. The partition is written to file `filename`, which should have a file type of `.par`. Note that the same partition definition can be used on different occasions to create partitions with different names. If the boolean argument `overwrite` is specified as `true`, and the specified export file already exists, it will be overwritten. If the argument is `false`, or is not given, then the perm will not overwrite the file, but will return an error message.

open(partition file)

Declaration

```dxl
PartitionFile open(string filename)
```

Operation

Creates a partition file and returns a handle. The file type must be `.par`. The handle can be used with dot notation to extract any of the properties available from a variable of type `PartitionFile`.

Example

This example checks that the file is a valid partition file:
```dxl
PartitionFile pf = open("partition.par")
```

close(partition file)

Declaration

```dxl
string close(PartitionFile pf)
```

Operation

Closes a partition file and releases the handle. If successful, returns a null string; otherwise returns a string containing an error message.
acceptReport

Declaration
string acceptReport(PartitionFile pf,
    string foldername)

Operation
Returns a string containing a report on information that would be produced if the partition in \textit{pf} is accepted into folder \textit{foldername}. This includes the names of the modules, attributes, and views which would be created.

acceptPartition

Declaration
string acceptPartition(Project p,
    PartitionFile pf,
    folder foldername)

Operation
Accepts the partition in \textit{pf} into folder \textit{foldername} in project \textit{p}.
If successful, returns a null string; otherwise returns a string containing an error message.

returnPartition

Declaration
string returnPartition(Project p,
    string partName,
    string returnDesc,
    string partFileName,
    bool isFinal,
    bool deleteData[, bool overwrite])

Operation
Returns the accepted partition with name \textit{partName}, using the description \textit{returnDesc}. This creates file \textit{partFileName}.
If \textit{isFinal} is \texttt{true}, the return is a final return: the data cannot be returned again. If \textit{isFinal} is \texttt{false}, the return is a synchronize operation, and the value of \textit{deleteData} is ignored.
If \textit{deleteData} is \texttt{true}, the return operation deletes all accepted data. If \textit{deleteData} is \texttt{false}, the return operation removes partition locks on the data, so that it remains in the database but is no longer partitioned in.
If the argument \textit{overwrite} is specified as \texttt{true}, and the specified file already exists, it will be overwritten. If the argument is \texttt{false}, or is not given, then the perm will not overwrite the file, but will return an error message.
If successful, returns a null string; otherwise returns a string containing an error message.
**rejoinReport**

**Declaration**

```c
string rejoinReport(PartitionFile pf,
                   string pathname)
```

**Operation**

Returns a string containing a report on information that would be produced if the partition in `pf` is rejoined. This includes the names of the modules, attributes, and views which would be created. The `pathname` argument is reserved for future enhancements; currently, it is ignored.

**rejoinPartition**

**Declaration**

```c
string rejoinPartition(Project p,
                        PartitionFile pf)
```

**Operation**

Rejoins the partition in `pf` into folder `foldername` in project `p`.

If successful, returns a null string; otherwise returns a string containing an error message.

**removePartition**

**Declaration**

```c
string removePartition(Project p,
                       string partName)
```

**Operation**

Recovers the information exported in `partName`, which must be the name of a partition exported from project `p`. This removes its partitioned out status, which enables it to be edited. Once removed, the partition can never be rejoined.

If successful, returns a null string; otherwise returns a string containing an error message.

**Partition information**

This section describes the functions and properties that allow access to the attributes of partitions and partition definitions. Some functions use the data type `PartitionPermission`, which has the same range of values as `Permission`, but applies only to data in partition definitions. This is the data type that confers the maximum access rights for users at the away database, if the partition definition is used to create a partition.
Partition properties

Partition properties are defined for use with the . (dot) operator and a partition handle to extract information from a partition or partition definition, as shown in the following syntax:

```
variable.property
```

where:

```
variable          is a variable of type PartitionDefinition,
                 PartitionModule, PartitionAttribute,
                 PartitionView, PartitionFile, InPartition, or
                 OutPartition.
```

The properties available vary according to the type being examined.

The types PartitionDefinition, PartitionModule, PartitionAttribute, and PartitionView refer to information in a partition definition.

<table>
<thead>
<tr>
<th>You can obtain an object of type</th>
<th>Using</th>
</tr>
</thead>
<tbody>
<tr>
<td>PartitionDefinition</td>
<td>load(partition definition) function</td>
</tr>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td></td>
<td>for partition definition in project loop.</td>
</tr>
<tr>
<td>PartitionModule</td>
<td>findModule function</td>
</tr>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td></td>
<td>for partition module in partition definition loop</td>
</tr>
<tr>
<td>PartitionAttribute</td>
<td>findAttribute function</td>
</tr>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td></td>
<td>for partition attribute in partition module loop</td>
</tr>
<tr>
<td>PartitionView</td>
<td>findView function</td>
</tr>
<tr>
<td></td>
<td>or</td>
</tr>
<tr>
<td></td>
<td>for partition view in partition module loop</td>
</tr>
<tr>
<td>PartitionFile</td>
<td>open(partition file) function</td>
</tr>
<tr>
<td></td>
<td>An object of type PartitionFile is created after a user at a home database has exported a partition and created a partition file.</td>
</tr>
<tr>
<td>OutPartition</td>
<td>for out-partition in project loop</td>
</tr>
<tr>
<td></td>
<td>An object of type OutPartition is created after a user at a home database has exported a partition definition. You can only access a type OutPartition using this loop.</td>
</tr>
</tbody>
</table>
You can obtain an object of type `InPartition` for in-partition in project loop.

An object of type `InPartition` is created after a user at a home database has exported a partition definition. You can only access a type `InPartition` using this loop.

### Partition definition properties

After a user at the home database has created a partition definition you can use these properties on a variable of type `PartitionDefinition`.

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Description of partition definition</td>
</tr>
<tr>
<td>name</td>
<td>Name of partition definition</td>
</tr>
</tbody>
</table>

### Partition module properties

After a user at the home database has created a partition definition you can use this property on a variable of type `PartitionModule`.

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of partition module</td>
</tr>
</tbody>
</table>

### Partition attribute properties

After a user at the home database has created a partition definition you can use this property on a variable of type `PartitionAttribute`.

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of partition attribute</td>
</tr>
</tbody>
</table>

### Partition view properties

After a user at the home database has created a partition definition you can use this property on a variable of type `PartitionView`.

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of partition view</td>
</tr>
</tbody>
</table>
Partition file properties

After a user at the home database has exported a partition and created a partition file, or after a user at the away database has synchronized or returned a partition, you can use these properties on a variable of type `PartitionFile`.

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>author</td>
<td>The user who created the partition file</td>
</tr>
<tr>
<td>date</td>
<td>Date the partition file was created</td>
</tr>
<tr>
<td>definitionName</td>
<td>Name of partition definition</td>
</tr>
<tr>
<td>description</td>
<td>Description of partition contained in file</td>
</tr>
<tr>
<td>name</td>
<td>Name of partition contained in file</td>
</tr>
<tr>
<td>subtype</td>
<td>If the type is Initial, returns &quot;ReadOnly&quot; if the file contains a partition in which all the data is read-only. Otherwise, returns &quot;Writeable&quot;. If the type is Final, returns &quot;Final&quot; if the file contains a partition that has been returned for the last time (not synchronized). Otherwise, if the file is a synchronize file, returns &quot;Intermediate&quot;.</td>
</tr>
<tr>
<td>timestamp</td>
<td>Timestamp of partition file</td>
</tr>
<tr>
<td>type</td>
<td>Returns &quot;Initial&quot; if the file contains a partition that is yet to be imported into the away database. Returns &quot;Final&quot; if the file contains a partition that has been returned or synchronized from the away database, and which should be rejoined or synchronized at the home database.</td>
</tr>
</tbody>
</table>

Out-partition properties

After a partition has been exported, the user at the home database can use these properties on a variable of type `OutPartition`.

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>author</td>
<td>The user who exported the partition</td>
</tr>
<tr>
<td>applyDate</td>
<td>Date the partition was exported</td>
</tr>
<tr>
<td>definitionName</td>
<td>Name of partition definition</td>
</tr>
<tr>
<td>description</td>
<td>Description of partition</td>
</tr>
<tr>
<td>folderName</td>
<td>Folder that contains all of the modules included in the partition definition</td>
</tr>
</tbody>
</table>
In-partition properties

After a partition has been imported, the user at the away database can use these properties on a variable of type InPartition.

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of partition</td>
</tr>
<tr>
<td>rejoinedBy</td>
<td>User who rejoined the partition</td>
</tr>
<tr>
<td>rejoinedDate</td>
<td>Date the partition was rejoined</td>
</tr>
</tbody>
</table>

In-partition properties

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceptDate</td>
<td>Date the partition was imported to the away database</td>
</tr>
<tr>
<td>applyDate</td>
<td>Date the partition was exported from the home database</td>
</tr>
<tr>
<td>author</td>
<td>The user who created the partition file</td>
</tr>
<tr>
<td>definitionName</td>
<td>Name of partition definition</td>
</tr>
<tr>
<td>description</td>
<td>Description of partition</td>
</tr>
<tr>
<td>folderName</td>
<td>Folder the partition was accepted into</td>
</tr>
<tr>
<td>name</td>
<td>Name of partition</td>
</tr>
<tr>
<td>returnedBy</td>
<td>User who returned the partition</td>
</tr>
<tr>
<td>returnedDate</td>
<td>Date the partition was returned</td>
</tr>
<tr>
<td>type</td>
<td>If the partition contains writable data, returns &quot;Writeable&quot;; otherwise, returns &quot;Read Only&quot;</td>
</tr>
</tbody>
</table>

for in-partition in project

Syntax

for inPartition in project do {
    ...
}

where:

    inPartition is a variable of type InPartition

    project is a variable of type Project
Operation

Assigns outPartition to be each successive imported partition record in the specified project. This is primarily for use in the away database.

for out-partition in project

Syntax

for outPartition in project do {
    ...
}
where:

outPartition is a variable of type OutPartition
project is a variable of type Project

Operation

Assigns outPartition to be each successive exported partition record in the specified project. This is primarily for use in the home database.

for partition definition in project

Syntax

for partDefinition in project do {
    ...
}
where:

partDefinition is a variable of type PartitionDefinition
project is a variable of type Project

Operation

Assigns partitionDefinition to be each successive partition definition within the specified project.

Partition access

This section describes the functions and properties that manage the partition and rejoin access rights.
isPartitionedOut, isPartitionedOutDef, isPartitionedOutVal

Declaration

string isPartitionedOut({Item i|Folder f|Project p| Module m|Object o|View v},
    bool &result)

string isPartitionedOut(AttrType at,
    bool &result)

string isPartitionedOutDef(AttrDef ad,
    bool &result)

string isPartitionedOutVal(AttrDef ad,
    bool &result)

Operation

If the current user has read access to the entity identified by the argument, sets result to indicate whether the entity is partitioned out, and returns a null string. If the current user does not have read access, returns an error message.

getPartitionMask, getPartitionMaskDef, getPartitionMaskVal

Declaration

string getPartitionMask({Item i|Folder f|Project p| Module m|Object o|View v},
    Permission &p)

string getPartitionMask(AttrType at,
    Permission &p)

string getPartitionMaskDef(AttrDef ad,
    Permission &p)

string getPartitionMaskVal(AttrDef ad,
    Permission &p)

Operation

This perm should only be used in the away database.

If the current user has read access to the entity identified by the argument, sets p to a mask of the entity’s permissions, and returns a null string. The mask describes the maximum access allowed to users in the away database. If the current user does not have read access, returns an error message.

If the data is partitioned in, the mask passed back is a bitwise OR of read, create, modify, and delete, access rights. If the data is not partitioned in, the mask is null.
Chapter 27
Requirements Interchange Format (RIF)

This chapter describes features that operate on Rational DOORS Requirements Interchange Format (RIF):

- RIF export
- RIF import
- RIF ID
- Merge
- RIF definition
- Examples

RIF export

exportType

Declaration
void initRIFExport(ExportType)

Operation
Sets the export type to be either RIF or ReqIF. Call this before using the exportPackage method.

example
initRIFExport(exportRIF_1_2)// Sets the export to be RIF
initRIFExport(exportReqIF)// Sets the export to be ReqIF

exportPackage

Declaration
string exportPackage(RifDefinition def, Stream RifFile, DB parent, bool& cancel)

Operation
Exports def to the XML file identified by RifFile. The stream must be have been opened for writing using “write (filename, CP_UTF8)”. If parent is null then a non-interactive operation is performed. Otherwise, progress bars will be displayed.

If an interactive export is performed, and is cancelled by the user, cancel will be set to true.
RIF import

importRifFile

Declaration

string importRifFile(string RifFilename, Folder parent, string targetName, string targetDesc, string RifDefName, string RifDefDescription, DB parent)

Operation

Performs a non-interactive import of RifFilename, placing the imported modules in a new folder in the specified parent. The new folder name and description are specified by targetName and targetDesc.

RifImport

A RifImport is an object which contains information on a RIF import. These are created by import operations, and are persisted in a list in the stored RifDefinition.

Properties are defined for use with the . (dot) operator and a RifImport handle to extract information from, or specify information in an import record, as shown in the following syntax:

variable.property

where:

variable is a variable of type RifImport.

property is one of the properties.

The following tables list the RifImport properties and the information they extract or specify:

<table>
<thead>
<tr>
<th>bool property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>mergeStarted</td>
<td>Returns true when a merge operation is started.</td>
</tr>
<tr>
<td>mergeCompleted</td>
<td>Returns true when the merge has been completed.</td>
</tr>
<tr>
<td>mergeRequired</td>
<td>Returns true when an import is a valid candidate for merging.</td>
</tr>
<tr>
<td>mergeDisabled</td>
<td>Returns true if the merge has been disabled due to lock removal.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>importedBy</td>
<td>Returns the user who performed the import.</td>
</tr>
<tr>
<td>mergedBy</td>
<td>Returns the user who performed the merge.</td>
</tr>
</tbody>
</table>
### Folder property

<table>
<thead>
<tr>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the folder containing the imported data. On import, a DXL script is expected to iterate through the contents of this folder, merging all items which have RIF IDs, and which are persisted in this folder.</td>
</tr>
</tbody>
</table>

### Date property

<table>
<thead>
<tr>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the time the export was performed. Note that this is the timestamp derived from the creationTime element of the header in the imported RIF package. Merges should be performed in the order in which the data was exported, rather than the order in which the packages were imported.</td>
</tr>
<tr>
<td>Returns the date that the import folder was created.</td>
</tr>
<tr>
<td>Returns the date that the merge of the import folder was completed, or started if it has not yet been completed.</td>
</tr>
</tbody>
</table>

---

### RIF ID

**getRifID**

**Declaration**

```dxl
string getRifID(Object o)
```

**Operation**

Returns a string with the RIF ID for object `o`. If the object does not have a RIF ID, an empty string is returned.

**getObjectByRifID**

**Declaration**

```dxl
Object getObjectByRifID(Module m, string s)
```

**Operation**

Returns the object within module `m` with a RIF ID of `s`. If the module does not contain an object with the input RIF ID, null is returned.
Merge

rifMerge

Declaration
string rifMerge(RifImport mrgObj, DB parent)

Operation
Performs a non-interactive merge using the information in mrgObj.

RIF definition

RifDefinition

A RifDefinition is the object in which a package to be exported in RIF format is defined.

Properties are defined for use with the . (dot) operator and a RifDefinition handle to extract information from a definition, as shown in the following syntax:

variable.property

where:

variable is a variable of type RifDefinition.

property is one of the following properties.

The following tables list the RifDefinition properties and the information they extract or specify.

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the definition.</td>
</tr>
<tr>
<td>description</td>
<td>The description of the definition.</td>
</tr>
<tr>
<td>rifDefinitionIdentifier</td>
<td>The unique ID of the RIF definition (this is shared between databases, unlike the name and description).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boolean property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>createdLocally</td>
<td>Returns true if the definition was created in the local database, as opposed to being imported.</td>
</tr>
</tbody>
</table>
RifModuleDefinition

A `RifModuleDefinition` is an object which contains the details of how a module should be exported, as part of a RIF package.

Properties are defined for use with the `.` (dot) operator and `RifModuleDefinition` handle to extract information from a definition record, as shown in the following syntax:

```
variable.property
```

where:

* `variable` is a variable of type `RifModuleDefinition`.
* `property` is one of the properties below.

The following tables list the `RifModuleDefinition` properties and the information they extract or specify:

<table>
<thead>
<tr>
<th>boolean property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>canModify</td>
<td>Returns true if the correct user can modify the definition.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>project</td>
<td>The project which contains the definition.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataConfigView</td>
<td>The name of the view used to define which data in the module will be included in the RIF export.</td>
</tr>
<tr>
<td>ddcView</td>
<td>The name of the view used to define what data can be edited when the exported RIF package is imported into another database.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bool property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>createdLocally</td>
<td>Whether the module was added to the RifDefinition in the current database or not.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ModuleVersion property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleVersion</td>
<td>The ModuleVersion reference for the given <code>RifModuleDefinition</code>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ddcmode property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddcMode</td>
<td>The type of access control used to define whether the module, or its contents, will be editable in each database once it has been exported.</td>
</tr>
</tbody>
</table>
**DdcMode constants**

DdcMode constants define the type of access control used define whether a module, or its contents, will be editable in each of the local and target database once the export has taken place. The following table details the possible values, and their meanings.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddcNone</td>
<td>Module will be editable in both source and target databases.</td>
</tr>
<tr>
<td>ddcReadOnly</td>
<td>Module will be editable in only the source database.</td>
</tr>
<tr>
<td>ddcByObject</td>
<td>Selected objects in the module will be made read-only in the source database.</td>
</tr>
<tr>
<td>ddcByAttribute</td>
<td>Selected attributes in the module will be made read-only in the source database.</td>
</tr>
<tr>
<td>ddcFullModule</td>
<td>Module will not be editable.</td>
</tr>
</tbody>
</table>

**for RifDefinition in Project**

**Syntax**

```dxl
for rifDef in proj do {
  ...
}
```

**Operation**

Assigns `rifDef` to be each successive `RifDefinition` in `Project proj`.

**for RifModuleDefinition in RifDefinition**

**Syntax**

```dxl
for rifModDef in rifDef so {
  ...
}
```

**Operation**

Assigns `rifModDef` to be each successive `RifModuleDefinition` in `RifDefinition rifDef`. 
for RifImport in RifDefinition

Syntax

for rifImp in rifDef do {
...
}

Operation

Assigns rifImp to be each successive rifImport in RifDefinition rifDef.

Examples

Example 1

This example dumps all information about all RIF definitions in the current project to the screen. It then conditional exports one of the packages.

RifDefinition rd
RifModuleDefinition rmd
Stream stm = write ("C:\\Public\\rifExport.xml", CP_UTF8)
string s = ""
bool b
Project p = current
Project p2
ModuleVersion mv
DB myDB = null
DdcMode ddcm

for rd in p do {

    print rd.name "\n"
    print rd.description "\n"
    print rd.rifDefinitionIdentifier "\n"

    if (rd.createdLocally) {

}
print "Local DB\n"
}

if (rd.canModify) {
    print "May be modified by current user\n"
}

p2 = rd.project

print fullName p "\n"

for rmd in rd do {
    print "\nModules present in definition :\n"

    mv = rmd.moduleVersion
    print fullName mv "\t"

    print rmd.dataConfigView "\t"
    print rmd.ddcView "\t"

    if (rmd.createdLocally) {
        print "Home DB.\n"
    }

    ddcm = rmd.ddcMode

    if (ddcm == ddcFullModule){
        print "Module will not be editable once definition is exported.\n"
} else if (ddcm == ddcByObject){

    print "Selected objects will be locked in the local database once the
definition is exported.\n"

} else if (ddcm == ddcByAttribute){

    print "Selected attributes will be locked in the local database once
the definition is exported.\n"

} else if (ddcm == ddcReadOnly){

    print "Module will only be editable in the local database once
definition is exported.\n"

} else if (ddcm == ddcNone){

    print "Module will be fully editable in both local and target
databases when definition is exported.\n"

}

if (rd.name == "RifDef1"){

    s = exportPackage (rd, stm, myDB, b)

    if (s != ""){

        print "Error occurred : " s "\n"
    }
}
}
Example 2

This example dumps all information about all RIF imports in the current project. It then merges those imports where required.

```
RifImport ri
RifDefinition rd
Project p = current
User importer, merger
string importerName, mergerName, res
Folder f
Skip dates = create

for rd in p do {
    for ri in rd do {

        rd = ri.definition
        print rd.name "\n"

        f = ri.folder
        print "Located in : " fullName f
        print "\n"

        importer = ri.importedBy
        importerName = importer.name
        print "Imported by : " importerName "\n"

        print "Imported on : " ri.importTime "\n"

        if (ri.mergeStarted && !ri.mergeCompleted) {
            print "Merge started on : " ri.mergeTime "\n"
        }
    }
}
```
} else if (ri.mergeCompleted) {

    print "Merge completed on : " ri.mergeTime "\n"

}

if (ri.mergeRequired) {

    print "Merge required.\n"
    res = rifMerge (ri, null)
    print "Merging result : " res "\n"

} else {

    merger = ri.mergedBy
    print "Merged by : " mergerName "\n"
}

if (ri.mergeDisabled) {

    print "Merge disabled, locks removed.\n"
}
print "\n"
Chapter 28

OLE objects

This chapter provides information on Rational DOORS DXL support for OLE technology. These functions are currently only available on Windows platforms. OLE technology support encompasses the linking and embedding of OLE objects and the use of the system clipboard to manipulate objects that can be embedded and linked to and from. OLE DXL supports automation with Rational DOORS as either client or server.

- Embedded OLE objects and the OLE clipboard
- OLE information functions
- Picture object support
- Automation client support
- Controlling Rational DOORS from applications that support automation

Embedded OLE objects and the OLE clipboard

This section defines DXL functions that allow OLE objects to be manipulated within Rational DOORS, and provide a programmatic means of controlling the OLE clipboard.

oleActivate

Declaration

bool oleActivate(Object o)
string oleActivate(Object o, Column c, integer index)

Operation

The first form activates the first OLE object embedded in the object text of o. The function returns true if the object text of o contains an OLE object and the activation of that object succeeds. Otherwise, it returns false.

The second form activates the OLE object at position index in the column c, for the object o.

The command uses the OLE object’s primary verb. For example, a Word object chooses to open in edit mode, while a video object chooses to play.

Example

/*
   this code segment checks whether the object text of the current formal object contains an OLE object, and if so, activates the first one.
*/
Object obj = current
if (oleIsObject obj) {
    if (oleActivate obj == false) {
        print "Problem trying to activate object\n"
    }
} else {
    print "Does not contain an embedded object in its object text\n"
}

/*
   this DXL script activates the second OLE object that exists in column 1 of the module display
*/
oleActivate(current Object, column 1, 1)

oleDeactivate

Declaration
bool oleDeactivate(Object o)
bool oleDeactivate(Object o, Column col, int oleIndex)

Operation
Deactivates the OLE object embedded o. The function returns true if o contains an activated OLE object and the deactivation succeeds. Otherwise, it returns false.

The second variant of this perm deactivates the OLE object specified by oleIndex in the specified column of the passed formal object. If the oleGetAutoObject() function was called to get the object’s dispatch pointer, the oleCloseAutoObject() function must be called to release the dispatch pointer before calling this function.

Example

/*
   this code segment checks whether the current formal object contains an OLE object in its object text, and if so, deactivates it
*/
Object obj = current
if (oleIsObject obj) {
    if (oleDeactivate obj == false) {
        print "Problem trying to deactivate object\n"
    }
} else {
    print "Does not contain an embedded object\n"
}
oleCopy

Declaration

bool oleCopy(EmbeddedOleObject oleObject)
string oleCopy(Object o, Column c, integer index)

Operation

The first form copies the embedded OLE object oleObject into the system clipboard. The OLE object can then be pasted into another Rational DOORS formal object or into any other Windows application that supports automation.

The second form copies the embedded OLE object at position index in column c for object o, into the system clipboard. The OLE object can then be pasted into another Rational DOORS formal object or into any other Windows application that supports automation.

Examples

void checkOLECopy(Object o, string attributeName)
{
    RichText rtf
    string s = richTextWithOle o.attributeName
    for rtf in s do
    {
        if (rtf.isOle)
        {
            EmbeddedOleObject ole = rtf.getEmbeddedOle
            oleCopy(ole)
            break
        }
    }
}
checkOLECopy(current Object, "Object Text")
/*
   this example copies the first OLE object in
   the current object, in column 1.
*/
string s = oleCopy(current Object, column 1, 0)
oleCut

Declaration

string oleCut(Object o, Column c, integer index)
bool oleCut(Object o)

Operation

The first form cuts the embedded OLE object at position index in column c for object o, into the system clipboard.
The OLE object can then be pasted into another Rational DOORS formal object or into any other Windows application that supports automation.

The second form cuts the embedded OLE object o into the system clipboard. The OLE object can then be pasted into another Rational DOORS formal object or into any other Windows application that supports automation.

The function returns true if o contains an OLE object and the cut operation succeeds. Otherwise, it returns false.

Example

/*
 this code segment checks whether the current formal object
 contains an OLE object in its object text, and if it so, cuts it to the
 system clipboard, and then pastes it into the next formal
 object in the current formal module
 */
Object obj = current
if (oleIsObject obj){
  if (oleCut obj){
    obj = next current
    if (obj != null){
      if (olePaste obj == false)
        print "Problem trying to paste object\n"
    }
  } else {
    print "Problem trying to cut object\n"
  }
} else {
  print "Does not contain an embedded object in its object text\n"
}
/*
 this DXL script cuts the second OLE object that exists in
 column 1 of the module display
 */
string s = oleCut(current Object, column 1, 0)
**oleDelete**

**Declaration**

bool oleDelete(Object o)

string oleDelete(Object o, Column c, integer index)

**Operation**

The first form removes the embedded OLE object from the object text of o. The function returns true if the object text of o contains an OLE object and the removal of that object succeeds. Otherwise, it returns false.

The second form deletes the OLE object in column c, for object o, at the index index.

**Example**

/*
this code segment removes an embedded OLE object from the object text of the current formal object.
*/

oleDelete (current Object)

**oleInsert**

**Declaration**

bool oleInsert(Object o,[attrRef],string fileName,[bool insertAsIcon])

where the optional parameter attrRef is in the following format: (Object o).(string attrName)

**Operation**

Embeds the file fileName as an OLE object in the Rational DOORS formal object o in a text attribute. If the optional parameter attrRef is specified, then the OLE object is embedded in the user-defined text attribute. If no parameter is specified, then the OLE object is embedded in the system Object Text attribute.

If the optional parameter insertAsIcon is specified, then if true, the OLE object is displayed as an icon, else it is displayed as content. If no parameter is specified, then the default is to display the OLE object as content.

The function returns true on successful insertion of the OLE object. Otherwise, it returns false.

An OLE package is created if a file has no associated applications that support OLE. OLE packages even allow executable files to be embedded into documents. It is then possible to execute such a file from within the document.

**Example**

/*
this code segment embeds an existing word document into the current formal object
*/

string docName = "c:\\docs\\details.doc"
Object obj = current
if (oleInsert(obj, obj."my_text", docName)){
   print "Successfully embedded document\n"
} else {
   print "Problem trying to embed document\n"
}

oleInsert (insert to buffer)

Declaration

bool oleInsert(Buffer bufInsert, string fileName, int charPos)

where:

- bufInsert is the buffer to insert the OLE into.
- fileName is the full filename of the file to insert as an OLE object. (The registered application for the file type must be available.)
- charPos is the offset in characters at which to insert the OLE. Any value less than 0 means insert at the end of the buffer.

Operation

Inserts OLE into the given buffer at a given character offset.

Example

Buffer myBuff = create
if (oleInsert(myBuff, "C:\example.pdf", -1))
   print (tempStringOf(myBuff)) "\n"
else
   print "Error!\n"
delete myBuff

oleIsObject

Declaration

bool oleIsObject(Object o)

Operation

Returns true if o contains an embedded OLE object in its Object Text attribute; otherwise, returns false.
Example

/*
 * this code segment checks to whether the current formal object contains an OLE object in its Object Text attribute, and if it does not, embeds a word document.
 */

string docName = "c:\\docs\\details.doc"
Object obj = current
if (oleIsObject obj){
    print "Already contains embedded object\n"
} else {
    oleInsert(obj, docName )
}

oleCloseAutoObject

Declaration

void oleCloseAutoObject(OleAutoObj &oa)

Operation

Closes an open OLE handle (interface) and deallocates the memory associated with it. It also sets the argument passed to it to null.

This function is useful for releasing handles that have been allocated, for example, through the oleGetAutoObject function. These handles are not normally released until the DXL program exits.

oleCloseAutoObject

Declaration

void oleCloseAutoObject(OleAutoObj &oa)

Operation

Closes an open OLE handle (interface) and deallocates the memory associated with it. It also sets the argument passed to it to null.

This function is useful for releasing handles that have been allocated, for example, through the oleGetAutoObject function. These handles are not normally released until the DXL program exits.

oleRTF

Declaration

Buffer oleRTF(EmbeddedOleObject, Buffer&)
Operation
Takes a chunk of richtext containing an OLE object, and returns the data as RTF loaded into the supplied buffer. This buffer is also returned allowing it to be used as an immediate assignment.
The buffer is emptied before the RTF is loaded.

olePaste

Declaration
bool olePaste(Object o)

Operation
Pastes the contents of the system clipboard into the object text of o as an embedded OLE object. The function returns true if o does not contain an OLE object and the paste operation succeeds. Otherwise, it returns false.

Example
/ * 
this code segment checks whether the current formal object contains an OLE object in its object text, and if it so, cuts it to the system clipboard, and then pastes it into the next formal object in the current formal module */
Object obj = current
if (oleIsObject obj){
    if (oleCut obj){
        obj = next current
        if (obj != null){
            if (olePaste obj == false){
                print "Problem trying to paste
                object\n"
            }
        }
    } else {
        print "Problem trying to cut object\n"
    }
} else {
    print "Does not contain an embedded object in its object text\n"
}

olePasteSpecial

Declaration
string olePasteSpecial(attrRef, bool displayAsIcon)
where attrRef can be one of:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

**Operation**

Copies an OLE object from the clipboard and appends it to `attRef`. The boolean `displayAsIcon`, when set to `true` will display the OLE object as an icon in the object. Returns null on success and displays an error message on failure.

**Example**

Object o = current
olePasteSpecial(o."object text", false)

---

### olePasteLink

**Declaration**

```dart
bool olePasteLink(Object o)
bool olePasteLink(attrRef)
```

where `attrRef` can be one of:

- (Object o).(string attrName)
- (Module m).(string attrName)
- (Link l).(string attrName)

**Operation**

The first form pastes the contents of the system clipboard `o` as a link to an OLE object. This function only succeeds if there is enough information about the data in the system clipboard to describe its location. Typically this function is used to link to a section of data in a larger body of data, for example, a paragraph in a Word document. The function returns `true` if `o` does not contain an OLE object and the paste operation succeeds. Otherwise, it returns `false`.

The second form inserts from the system clipboard into the text attribute referred to by `attrRef`.

**Example**

```dart
/*
this code segment checks to see whether the current formal object contains an
OLE object in its object text, and if it does not, pastes a link to the object
described in the system clipboard.
*/

Object obj = current
if (oleIsObject obj == false){
    if (olePasteLink obj == false){
        print "Problem trying to paste link to
object\n"
```
oleSaveBitmap

Declaration

oleSaveBitmap(Object o)

Operation

Forces a write of the picture for the current object. This affects OLE display on UNIX platforms.

Example

Object o = current
oleSaveBitmap (o)

oleCount

Declaration

int oleCount(attrRef)

where attrRef can be one of:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation

Returns the number of OLE objects embedded in the attribute (new version of oleIsObject(Object))

Example

Object o = current
int n = oleCount(o."Object Text")
print "Number of OLE objects in Object Text attribute for current object: " n ""
**Operation**

Returns *true* if an OLE object is selected in the specified Object *o*. If anything other than an OLE object is selected (e.g. text and an OLE object), the function returns *false*. If two or more contiguous OLE objects are selected, the function returns *true*.

---

**showOlePropertiesDialog**

**Declaration**

```java
void showOlePropertiesDialog(Object o)
```

**Operation**

Shows the OLE properties dialog for the selected OLE object in the specified Object *o*.

- If no OLE object is selected, the dialog will not appear.
- If anything other than an OLE object is selected (e.g. text and an OLE object), the function returns *false*.
- If two or more contiguous OLE objects are selected, the options dialog will appear for the first object.

---

**containsOle**

**Declaration**

```java
bool containsOle(attrRef)
```

where `attrRef` can be one of:

- `(Object o).(string attrName)`
- `(Module m).(string attrName)`
- `(Link l).(string attrName)`

**Operation**

Returns *true* if the specified attribute contains OLE data

**Example**

```java
Object o = current
if (containsOle(o."Object Text")){
    oleActivate(o)
}
```
OLE information functions

**getOleWidthHeight**

**Declaration**

```
string getOleWidthHeight(EmbeddedOleObject embedOle, int &width, int &height)
```

**Operation**

These functions provide information on Embedded OLE objects as demonstrated by the following examples.

**Example 1**

```dxl
void checkOLEcount(Object o, string attributeName)
{
    int n = oleCount(o.attributeName)
    RichText rtf
    string s = richTextWithOle o.attributeName
    int j = 0
    for rtf in s do
    {
        if (rtf.isOle)
        {
            j++
        }
    }
    if (j != n)
    {
        print "ERROR: oleCount gives " n " and for rtf in string gives " j "\n"
    } else {
        print "OK: they both give " n "\n"
    }
}
```

Object o = current
checkOLEcount(o, "Object Text")
Example 2

```dxl
void checkExportPicture(Object o, string attributeName, string baseFileName)
{
    EmbeddedOleObject ole
    int i = 1
    string errmess = null
    RichText rtf
    string s = richTextWithOle o.attributeName
    i = 1
    for rtf in s do
    {
        if (rtf.isOle)
        {
            ole = rtf.getEmbeddedOle
            string filename = baseFileName "-rtfloop-" i ".png"
            print "Exporting " filename "\n"
            errmess = exportPicture(ole, filename , formatPNG)
            if (!null errmess)
            {
                print "ERROR: " errmess "\n"
            }
            i++
        }
    }
}
Object o = current
checkExportPicture(o, "Object Text", "C:\temp\")
```

Example 3

```dxl
void checkOLECopy(Object o, string attributeName)
{
    RichText rtf
    string s = richTextWithOle o.attributeName
    for rtf in s do
    ```
{ if (rtf.isOle)
{ EmbeddedOleObject ole = rtf.getEmbeddedOle
oleCopy(ole)
break
}
}

checkOLECopy(current Object, "Object Text")

Example 4

void checkOLEWidthHeight(Object o, string attributeName)
{
    EmbeddedOleObject ole
    RichText rtf
    string s = rtfWithOle o.attributeName
    int width, height
    for rtf in s do
    {
        if (rtf.isOle)
        {
            ole = rtf.getEmbeddedOle
            getOleWidthHeight(ole, width, height)
            print("width = " width ", height = " height "\n")
        }
    }
}

checkOLEWidthHeight(current Object, "Object Text")

//run with an object containing several OLEs of different sizes in the object text
Example 5

Object o = current
int width
int height
string mess = getPictWidthHeight(o, width, height)
if (null mess)
{
    print "w = " width ",  h = " height "\n"
}else{
    print mess "\n"
}

Run this against an object with an embedded picture, an object with at least one OLE object in the object text and an object with no OLE objects or pictures.

oleSetMaxWidth

Declaration

string oleSetMaxWidth(attrRef, int width)
where attrRef can be one of:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation

Sets the maximum width of an OLE object in the attribute attrRef. Any OLE object wider will be scaled down to fit the column (the aspect ratio will be maintained).

Returns an error message if anything goes wrong.

oleSetMinWidth

Declaration

string oleSetMinWidth(attrRef, int width)
where attrRef can be one of:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation
Sets the minimum width of an OLE object in the attribute attrRef. Any OLE object narrower will be scaled up to fit the column (the aspect ratio will be maintained).
Returns an error message if anything goes wrong.

oleSetHeightandWidth

Declaration
oleSetHeightandWidth(attrRef, int height, int width, int index)
where attrRef can be one of:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation
Sets the height and width of the OLE object within attrRef at the specified index.

Example
Object o = current Object
oleSetHeightandWidth(o."Object Text", 150, 150, 1)

oleResetSize

Declaration
string oleResetSize(attrRef)
where attrRef can be one of:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation
Resets the width and height of the OLE objects in the attribute attrRef to their actual size.

Example 1
Scale to fit main column
Sets the max and min width of OLEs in the object text to the width of the main column.
Object obj
Column col
Column mainColumn
for col in current Module do
{
    if (main col)
    {
        mainColumn = col
        break
    }
}
int width = width(mainColumn)
string errmess = ""
for obj in current Module do
{
    int numOles = oleCount(obj."Object Text")
    if (numOles > 0)
    {
        errmess = oleSetMaxWidth(obj."Object Text", width)
        if (!null errmess)
        {
            break
        }
        errmess = oleSetMinWidth(obj."Object Text", width)
        if (!null errmess)
        {
            break
        }
    }
}
if (!null errmess)
{
    print "ERROR:" errmess "\n"
}

Example 2
Reset main column OLEs
Resets the size of all OLE objects in the Object Text

Object obj
Column col
Column mainColumn
for col in current Module do
{
    if (main col)
    {
        mainColumn = col
        break
    }
}
int width = width(mainColumn)
string errmess = ""
for obj in current Module do
{
    int numOles = oleCount(obj."Object Text")
    if (numOles > 0)
    {
        errmess = oleResetSize(obj."Object Text")
        if (!null errmess){
            break
        }
    }
}
if (!null errmess)
{
    print "ERROR:" errmess "\n"
}
Picture object support

These functions allow DXL to import pictures into Rational DOORS, and work with objects that contain pictures. In Rational DOORS 6.0 and later, pictures are unique to an object, and it is not necessary to identify a picture with a name. The functions using a picture name argument should be used for backwards compatibility only.

Constants

The following constants of type `int` are valid values for arguments that specify the format of a picture:

<table>
<thead>
<tr>
<th>Import format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formatBMP</td>
<td>Bitmap</td>
</tr>
<tr>
<td>formatDIB</td>
<td>Bitmap</td>
</tr>
<tr>
<td>formatWMF</td>
<td>Windows meta file</td>
</tr>
<tr>
<td>formatEPSF</td>
<td>Encapsulated PostScript</td>
</tr>
<tr>
<td>formatUNKNOWN</td>
<td>Unknown format</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Export format</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>formatPNG</td>
<td></td>
</tr>
</tbody>
</table>

changePicture

Declaration

```dxl
bool changePicture(string currentName, string newName)
```

Operation

Changes the name of a given picture by passing the current and new names. Returns `true` if the operation succeeds. This function is retained only for compatibility with earlier releases.

copyPictureObject

Declaration

```dxl
void copyPictureObject(Object source, Object target)
```
Operation
Copies a picture from the source object to the target object. It generates a run-time DXL error if either argument is null.

deletePicture

Declaration
bool deletePicture(Object o)
bool deletePicture(string pictureName)

Operation
Deletes the picture in object o. If the object is not a picture, the call fails.
The second form is retained only for compatibility with earlier releases. All new programs should use the first form.

Example
if (deletePicture current) {
    print "Successful deletion\n"
} else {
    print "Failed to delete picture\n"
}

exportPicture

Declaration
string exportPicture(Object obj,
                     string fileName,
                     int format)

Operation
Exports a picture, including OLE objects, associated with a given object to the file fileName in the specified format.
Some pictures, when exported, may have a black border.
The only supported export format is formatPNG.

Example
Object o = current
string n = o."PictureName"
string s = exportPicture(o, n ".png", formatPNG)
if (!null s) {
    print s " : " n "\n"}
exportPicture

**Declaration**

```dxl
string exportPicture(EmbeddedOleObject oleObject,
                     string fileName,
                     int format)
```

**Operation**

Exports a picture, including OLE objects, associated with a given object to the file `fileName` in the specified format.

**Example**

```dxl
void checkExportPicture(Object o, string attributeName, string baseFileName)
{
    EmbeddedOleObject ole
    int i = 1
    string errmess = null
    RichText rtf
    string s = richTextWithOle o.attributeName
    i = 1
    for rtf in s do
    {
        if (rtf.isOle)
        {
            ole = rtf.getEmbeddedOle
            string filename = baseFileName "-rtfloop=" i ".png"
            print "Exporting " filename "\n"
            errmess = exportPicture(ole, filename , formatPNG)
            if (!null errmess)
            {
                print "ERROR: " errmess "\n"
            }
            i++
        }
    }
    Object o = current
    checkExportPicture(o, "Object Text", "C:\\temp\\")
```
getPictBB

Declaration

void getPictBB(Object o,
    int &llx,
    int &lly,
    int &urx,
    int &ury)

Operation

Returns the picture’s bounding box measured in tenths of a point. The bounding box is specified by its lower-left and upper-right co-ordinates.

gPictFormat

Declaration

string getPictFormat(Object o)
int getPictFormat(Object o)

Operation

The first form returns the name of the format of the picture in object o.
The second form returns an integer corresponding to the format of the picture in object o.

Possible format names and integers are:

<table>
<thead>
<tr>
<th>Format</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;EPSF&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;BMP&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;WMF&quot;</td>
<td>3</td>
</tr>
</tbody>
</table>

Example

if (getPictFormat current Object != "EPSF") {
    ack "Cannot output this picture format"
    halt
}

gPictName

Declaration

string getPictName(Object o)
**Operation**

If `o` contains a picture in a format supported by Rational DOORS, this function returns the picture file name; otherwise, it returns `null`. The returned file name should be treated as a read-only handle. This function is intended for use by exporters.

If the operation fails, returns `null`.

---

**getPictWidthHeight**

**Declaration**

```dxl
string getPictWidthHeight(Object o,  
    int &width,  
    int &height,)
```

**Operation**

On return, passes back by reference the picture’s width and height in pixels. The object must contain either a picture or an OLE object.

On Windows platforms, if it is an OLE object, a bitmap is generated of the OLE object, then the width and height taken of the bitmap.

On UNIX platforms, this function returns the width and height of the picture snapshot of the OLE object (picture snapshots are stored in the database if `oleunix=true` is included in the registry). If a snapshot does not exist, returns an error message to indicate that the width and height are unavailable.

If the operation succeeds, returns `null`; otherwise returns an error message.

**Example**

```dxl
int width
int height
Object o = current
bool bIsPicture = o."Picture"
bool bIsOLE = o."OLE"
if (bIsPicture || bIsOLE) {
    string errmsg = getPictWidthHeight(o)
    if (null errmsg) {
        print "width = " width " ,height=" height ""
    } else {
        print errmsg
    }
}
```
importPicture

Declaration
bool importPicture(string pictureName,
                   string fileName,
                   string format)

Operation
Imports pictures into Rational DOORS. This function is retained for compatibility with earlier releases, but is redundant in Rational DOORS 6.0 and later.

The pictureName argument is the name for the picture once it is imported; fileName is the file you are importing from; and format is the format of the picture, which can be one of "WMF", "BMP" or "EPSF" (case insensitive).

Returns true if the import succeeds; otherwise, returns false.

Example
if (importPicture("Test", "c:\test.bmp", "BMP")) {
  print "Successfully imported picture test.bmp\n"
} else {
  print "Failed to import picture test.bmp\n"
}

insertBitmapFromClipboard

Declaration
bool insertBitmapFromClipboard(Object insertHere)

Operation
Inserts a bitmap of any format except an OLE object from the Windows clipboard into the object insertHere. The object must already contain a picture, which is replaced. If the operation succeeds, returns true; otherwise, returns false. If insertHere is null, the call fails.

For UNIX platforms, returns false.

saveClipboardBitmapToFile

Declaration
bool saveClipboardBitmapToFile(string fileName)

Operation
If there is a valid bitmap on the Windows clipboard, saves it to the specified file. The argument fileName can be an absolute or relative path. If the operation succeeds, returns true; otherwise, returns false.
For UNIX platforms, returns false.

Example

```dxl
string FileName=tempFileName()
saveClipboardBitmapToFile(FileName)
```

---

**insertPictureAfter**

**Declaration**

```dxl
bool insertPictureAfter(string pictureName, Object insertHere)
```

**Operation**

Inserts picture `pictureName` after an object `insertHere`. This function is supported only for compatibility with earlier releases. In new programs, use the `insertPictureFileAfter` function.

**Example**

```dxl
if (insertPictureAfter("Test", current Object)) {
    print "Successful picture insertion\n"
} else {
    print "Failed to insert the picture\n"
}
```

---

**insertPictureBelow**

**Declaration**

```dxl
bool insertPictureBelow(string pictureName, Object insertHere)
```

**Operation**

Inserts picture `pictureName` below an object `insertHere`. This function is supported only for compatibility with earlier releases. In new programs, use the `insertPictureFileBelow` function.

**Example**

```dxl
if (insertPictureBelow("Test", current Object)) {
    print "Successful picture insertion\n"
} else {
    print "Failed to insert the picture\n"
}
```

---

**insertPictureFile**

**Declaration**

```dxl
bool insertPictureFile(string fileName,
                        int format
                        Object insertHere)
```
Operation

Inserts picture \textit{fileName} into object \textit{insertHere}, which must be a picture object. If the operation succeeds, an existing picture in the object is replaced with that in \textit{fileName}. The format argument can be one of the import values listed in “Constants,” on page 755.

Example

Object currentObject = current
bool Result = false
string BitmapFileName = "c:\\test.bmp"
if (currentObject == null) {
    //No objects currently exist in the module
    Result = insertPictureFile(BitmapFileName, formatBMP, currentObject)
} else {
    Result = insertPictureFile(BitmapFileName, formatBMP, currentObject)
}
if(Result) {
    print "Successful picture insertion\n"
}

\textbf{insertPictureFileAfter}

Declaration

\begin{verbatim}
bool insertPictureFileAfter(string fileName, int format
Object insertHere)
\end{verbatim}

Operation

Inserts picture \textit{fileName} after an object \textit{insertHere}. The format argument can be one of the import values listed in “Constants,” on page 755.

Example

Object currentObject = current
bool Result = false
string BitmapFileName = "c:\\test.bmp"
if (currentObject == null) {
    //No objects currently exist in the module
    Result = insertPictureFileAfter(BitmapFileName, formatBMP, null)
} else {
    Result = insertPictureFileAfter(BitmapFileName, formatBMP, currentObject)
if(Result) {
    print "Successful picture insertion\n"
}

---

**insertPictureFileBelow**

**Declaration**

```dxl
bool insertPictureFileBelow(string fileName, 
    int format 
    Object insertHere)
```

**Operation**

Inserts picture `fileName` below an object `insertHere`. The format argument can be one of the import values listed in “Constants,” on page 755.

**Example**

```dxl
Object currentObject = current 
bool Result = false 
string BitmapFileName = "c:\\test.bmp" 
if (currentObject == null) {
    //No objects currently exist in the module 
    Result = insertPictureFileBelow(BitmapFileName, 
        formatBMP,null)
} else {
    Result = insertPictureFileBelow(BitmapFileName, 
        formatBMP,currentObject)
}
if(Result) {
    print "Successful picture insertion\n"
}
```

---

**oleLoadBitmap**

**Declaration**

```dxl
Bitmap oleLoadBitmap(DBE dialog, 
    Object fromHere, 
    bool lockColors, 
    int& width, 
    int& height)
```

**Operation**

Returns a bitmap handle for the given OLE object, provided the OLE bitmap has been stored.
The handle to the bitmap can then be used to draw the picture onto a canvas.
The function requires passed width and height; when the function exits these become the width and height of the picture.
OLE bitmaps are only stored in Rational DOORS 4.1 and later releases, and then only if you have not run Rational DOORS with the command line option to prevent it storing the picture.

Example
def oleLoadBitmap(dbMain, current Object, true, w, h):
    DBE dbMain
    void doDraw(DBE dbMain) {
        Bitmap b
        int w, h
        b = oleLoadBitmap(dbMain, current Object, true, w, h)
        drawBitmap(dbMain, b, 0,0)
    }
    DB artBox = create "Try resizing"
    dbMain = canvas(artBox, 400, 300, doDraw)
    show artBox

openPictFile

Declaration
Stream openPictFile(Object o)

Operation
Opens a read-only stream for the file containing the picture referenced in the named object.

Example
Stream picture = openPictFile thisObj

pictureCopy

Declaration
bool pictureCopy(Object object)

Operation
On Windows platforms only, copies the picture in the specified object to the system clipboard.

Example
bool Result = pictureCopy(current)
```dxl
if (Result) {
    print "Picture successfully copied\n"
} else {
    print "Picture not copied\n"
}
```

**reimportPicture**

**Declaration**

```dxl
bool reimportPicture(string pictureName)
```

**Operation**

This function is provided only for compatibility with earlier releases. It has no effect in Rational DOORS 6.0 or later.

**for pictures in project**

**Syntax**

```dxl
for s in pictures(Project p) do {
    ...
}
```

**Operation**

This loop is retained for compatibility with earlier release. Because of the changes to access restrictions in Rational DOORS 6.0, where pictures are specific to a module, this loop returns the names of the pictures in the current module only.

**Example**

This example prints the names of all pictures in the current module.

```dxl
string s
for s in pictures current Project do {
    print s " \n"
}
```

**supportedPictureFormat**

**Declaration**

```dxl
bool supportedPictureFormat(int format)
```
Operation
Returns true if the specified format is supported by the current client.

Example
if(supportedPictureFormat(formatWMF)){
    print "WMF format is supported.\n"
}

pictureCompatible

Declaration
bool pictureCompatible(string filename, int format)

Operation
Returns true if the specified file has header information which indicates that it contains a picture of the specified format.

Example
string fileName = “C:\temp\mypic.bmp”
if(pictureCompatible(fileName, formatBMP)){
    print fileName " is a valid BMP file.\n"
}

Automation client support

This section defines DXL functions with which Rational DOORS can be used as an automation client. That means Rational DOORS can be used to control other Windows applications that provide automation interface objects. Information on interface objects, methods and properties for other applications is available in the relevant automation documentation.

The functions fall into three groups:

- Accessing an interface
  The DXL functions oleCreateAutoObject and oleGetAutoObject provide access to automation interfaces in other applications. In addition to obtaining interface objects in these specific ways, interface objects can also be retrieved by accessing the properties or making method calls to other interface objects.

- Getting and setting properties
  The DXL functions oleGet and olePut provide access to automation object properties. The values of a property can be retrieved from an automation object, and where the object enables it, they can also be set.

- Calling automation methods
In addition to providing access to properties, automation interfaces can also provide methods. These provide access to capability in the other application and can return data as a result of their execution. In addition they might require data to be passed to them as arguments. Rational DOORS provides support for automation methods with the various oleMethod functions, the OleAutoArgs variable type and the various functions that can be used to manipulate variables of that type: create(OleAutoArgs), delete(OleAutoArgs), clear(OleAutoArgs), put(OleAutoArgs), and oleMethod.

### oleGetResult

**Declaration**

```cpp
string oleGetResult()
```

**Operation**

Rational DOORS provides the read-write Result property to automation clients, enabling them to exchange information with DXL programs. This function gets the value of this property.

**Example**

```cpp
if (oleGetResult == "OK") {
    // operation was successful
}
```

### oleSetResult

**Declaration**

```cpp
void oleSetResult(string message)
```

**Operation**

Rational DOORS provides the read-write Result property to automation clients, enabling them to exchange information with DXL programs. This function sets the value of this property.

### oleCreateAutoObject

**Declaration**

```cpp
OleAutoObj oleCreateAutoObject(string autoObjName)
```

**Operation**

Obtains a reference to a named automation interface. With a type OleAutoObj it is then possible to access properties and call methods. The application to support the interface object is started when this function is called.

**Example**

```cpp
OleAutoObj theWordApp = oleGetAutoObject("Word.Application")
```
if (null theWordApp) {
    theWordApp = oleCreateAutoObject("Word.Application")
}
olePut (theWordApp, "visible", true)
infoBox "Now you see it."
olePut (theWordApp, "visible", false)
infoBox "Now you don't."

**oleGetAutoObject**

**Declaration**

OleAutoObj oleGetAutoObject(Object o)

OleAutoObj oleGetAutoObject(string autoObjName)

**Operation**

The first form obtains a reference to an automation interface object for the OLE object embedded in o. The OLE object must be activated using the oleActivate() function before calling this function. With a type OleAutoObj it is then possible to access properties and call methods. The application to support the interface object is started when this function is called.

This function returns the base level interface to the embedded object. Not all objects that support embedding and automation also support automation of embedded objects. For objects that are not supported, null is returned.

The second form obtains a reference to an instance of the application that is already running. For an example of its use, see the example for the oleCreateAutoObject function.

**oleGet**

**Declaration**

string oleGet(OleAutoObj autoObj,
              string propertyName,
              [OleAutoArgs argumentList,]
              {string|int|bool|char|OleAutoObj}
              &Result)

**Operation**

Obtains the value of a specified property for a specified automation object, with optionally a list of arguments of type OleAutoArgs, and with Result set to the appropriate type.

The variation of this function that enables access to an OleAutoObj value is particularly useful when controlling an application that has a hierarchy of objects.

If the value of a property is successfully returned, returns null; otherwise returns a string containing an error message.
Example
This example obtains a reference to an automation interface to Excel, gets the visible attribute, and makes it visible if it is hidden:

```c
OleAutoObj objExcel = oleCreateAutoObject("Excel.Application")
if (objExcel != null){
    bool excelVisible
    oleGet(objExcel, "Visible", excelVisible)
    if (excelVisible == false){
        olePut(objExcel, "Visible", true)
    }
}
```

__olePut__

**Declaration**
```
string olePut(OleAutoObj autoObj,
              string propertyName,
              {string|int|char|bool|OleAutoObj} newValue)
```

**Operation**
Sets the value of a specified property for a specified automation object, with `newValue` set to the appropriate type. If the value of a property is successfully set, returns `null`; otherwise, it returns a string containing an error message.

__create(OleAutoArgs)__

**Declaration**
```
OleAutoArgs create(void)
```

**Operation**
Initializes and returns a type `OleAutoArgs` variable.

**Example**
```
/*Typical call to create for an OleAutoArgs variable*/
OleAutoArgs autoArgs = create
```

__delete(OleAutoArgs)__

**Declaration**
```
void delete(OleAutoArgs autoArgs)
```
Operation
Destroys a type OleAutoArgs variable and frees any system resources used by it. After a type OleAutoArgs variable has been deleted with this function, it becomes invalid and cannot be used again until initialized with the create (OleAutoArgs) function.

Example
This example is a typical call to delete for a variable of type OleAutoArgs:
OleAutoArgs autoArgs = create
delete(autoArgs)

clear(OleAutoArgs)

Declaration
void clear(OleAutoArgs autoArgs)

Operation
Empties the contents of a type OleAutoArgs variable, returning it to the state it was in immediately after it was initialized with the create command. This enables a single type OleAutoArgs variable to be created and then reused again and again throughout a DXL application.

Example
This example is a typical call to clear for a variable of type OleAutoArgs:
OleAutoArgs autoArgs = create
clear(autoArgs)

/*
code using the same autoArgs variable for something different
*/
delete(autoArgs)

put(OleAutoArgs)

Declaration
void put(OleAutoArgs autoArgs,  
   [string argName,]  
   {string|int|char|bool|OleAutoObj} value)

Operation
Stores a value of the appropriate type in a type OleAutoArgs variable autoArgs. The optional argument argName enables arguments to be named. If it is omitted, the values are inserted into the argument block in the order in which they are supplied.
This means that where the automation object supports named arguments, the formal ordering of arguments is not necessary. Both named and ordered arguments are permitted in the same OleAutoArgs variable.

For examples of usage see the example for the oleMethod function.

### oleMethod

**Declaration**

```dxl
string oleMethod(OleAutoObj autoObj,
    string methodName
    [,OleAutoArgs autoArgs
    [,{string|int|char|bool|OleAutoObj}
    result]])
```

**Operation**

Uses a specific automation interface to call a specific automation method. Optionally you can specify an argument block. With an argument block, optionally, you can specify a return value of a specific type. If the operation succeeds, oleMethod returns null; otherwise, it returns a string containing an error message.

### Controlling Rational DOORS from applications that support automation

This section defines functions for controlling Rational DOORS from other applications that support automation. For example, Visual Basic macros can be created in Excel to send commands to Rational DOORS.

### Automation interface

Rational DOORS provides an automation interface for other applications to use to control Rational DOORS. This object is called DOORS.Application. It provides two methods that can be called from other applications, along with the property Result.

The property, DOORS.Application.Result, enables other applications to exchange information with Rational DOORS in both directions. From Rational DOORS, use the oleGetResult and oleSetResult functions to pass information to and from a Visual Basic program.

**Example**

```dxl
/*
This is an example of an Excel macro that calls Rational DOORS, logging in as user 'John Smith' with password 'password', and sets the result message for use with oleGetResult.
*/

Sub testDoors()
```
Set DOORSObj = CreateObject("DOORS.Application")
SendKeys "John Smith" & "{TAB}" & "password" & _"{ENTER}", True
DOORSObj.Result = "Just checking"
End Sub

Sub testDoors()
Set DOORSObj = CreateObject("DOORS.Application")
SendKeys "John Smith" & "{TAB}" & "password" & _"{ENTER}", True
DOORSObj.runFile ("c:\doors\lib\dxl\example\ddbintr0.dx1")
End Sub

(runStr sample)
Sub testDoors()
Set DOORSObj = CreateObject("DOORS.Application")
SendKeys "John Smith" & "{TAB}" & "password" & _"{ENTER}", True
DOORSObj.runStr ("current = create(""Demo"", ""Demo"", """, 1); Object o = create current Module; o.""Object Heading"" = ""From Excel via OLE""")
End Sub

---

**runFile**

**Syntax**

runFile(dx1FileName)

where:

*dx1FileName* is a full path

**Operation**

This method enables other applications to pass Rational DOORS the path and file name of a DXL file, then requests Rational DOORS to run it.

**Example**

This example is an Excel macro that calls Rational DOORS, logging in as user *John Smith* with password *password*, and requests it to run the *ddbintr0* example from the DXL library:
Sub testDoors()
    Set DOORSObj = CreateObject("DOORS.Application")
    SendKeys "John Smith" & "{TAB}" & "password" & "{ENTER}", True
    DOORSObj.runFile ("c:\doors\lib\dxl\example\ddbintro.dxl")
End Sub

Note: There has been a change in functionality between Rational DOORS 7.x and Rational DOORS 8 concerning runFile. Any files passed to runFile must be transcoded to UTF-8 encoding rather than Latin-1. Alternatively, you can use runStr to include a file. The behavior of runStr is unchanged since version 7.

---

### runStr

**Syntax**

runStr(dxlText)

where:

dxlText is a string

**Operation**

This method enables other applications to pass Rational DOORS a string containing DXL functions for Rational DOORS to execute.

You can send more than one line to runStr at a time by using \n or ; in the string.

**Example**

This example is an Excel macro that calls Rational DOORS, logging in as user John Smith with password password, and requests it to create a new module. The macro then creates an object in the module:

Sub testDoors()
    Set DOORSObj = CreateObject("DOORS.Application")
    SendKeys "John Smith" & "{TAB}" & "password" & "{ENTER}", True
    DOORSObj.runStr("current = create(""Demo"", ""Demo"", "", 1);
                   Object o = create first current Module;
                   o."Object Heading" = "From Excel via OLE"")
End Sub
Chapter 29

Triggers

This chapter describes triggers, a powerful mechanism for associating Rational DOORS scripts with events in Rational DOORS.

- Introduction to triggers
- Trigger constants
- Trigger definition
- Trigger manipulation
- Drag-and-drop trigger functions

Introduction to triggers

Triggers are a mechanism in DXL for associating an event, such as opening a project or modifying an attribute, with a DXL program. This provides a very powerful customization facility that can be used for a number of tasks, including process enforcement. Triggers are created, managed, and deleted in DXL.

Note: Rational DOORS Web Access does not support triggers.

There are two examples demonstrating the use of triggers: defview.dxl and delview.dxl that permit the automatic loading of a user’s preferred view in a formal module. These are in the directory called: $DOORSHOME/lib/dxl/example

Triggers are described in terms of level, type and event.

There are six trigger levels:
- module
- object
- attribute*
- links
- discussion
- comment

*The use of module-level system attributes, such as Name and Description, is not supported with triggers.

There is one trigger type:
- post

There are two trigger events:
A **pre-event** trigger is a mechanism for performing an action or a check before an event happens. The code executed can return a veto, which prevents the subsequent event from happening. When multiple triggers have been defined for the same event, trigger execution is ordered on the trigger’s priority. For a pre-event to succeed, all pre-events must succeed.

A **post-event** trigger is executed after the associated event happens, for example after a module is opened.

---

**Basic trigger events**

There are eight basic event types:

<table>
<thead>
<tr>
<th>Event</th>
<th>Synonyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>edit, read</td>
</tr>
<tr>
<td>close</td>
<td></td>
</tr>
<tr>
<td>save</td>
<td>write, modify</td>
</tr>
<tr>
<td>sync</td>
<td></td>
</tr>
<tr>
<td>drag</td>
<td></td>
</tr>
<tr>
<td>drop</td>
<td></td>
</tr>
<tr>
<td>create</td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td></td>
</tr>
</tbody>
</table>

The only current application for the sync event is changing the current object in a formal and link module. The following table shows currently supported event and level combinations:

<table>
<thead>
<tr>
<th>open</th>
<th>close</th>
<th>save</th>
<th>sync</th>
<th>drag</th>
<th>drop</th>
<th>create</th>
<th>delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Both pre-event and post-event types are supported for all marked combinations.

Trigger levels have the following two extra dimensions: **scope** and **priority**.

The object open event will only occur when double-clicking on the object. Viewing the object through Object Properties will not cause an open event.
Trigger scope

Triggers are database wide or specific to a module, object, or attribute. They can be generic or specific.

**generic**
Trigger applies to all entities at trigger level, for example, all modules, all objects, all attributes.

**specific**
Trigger applies to a specific entity, for example, module "URD" means the module called URD, current means the current entity (as in current Module) at a trigger level.

Specific items defined are for each level.

- **project**
  project name

- **module**
  module name

- **object**
  absolute number, as string

- **attribute**
  attribute name

Generic module triggers are stored in the specified project or the current project, unless the trigger is specified as database wide using the `project->all` syntax. In this case, they are stored in the database root folder.

**Trigger Example:**

Trigger "t1" applies to all modules in the database.

```dxl
trigger("t1", project->all->module->all, pre, open, 10, …)
```

Trigger "t2" applies to all modules in project "p1".

```dxl
trigger("t2", project->"p1"->module->all, pre, open, 10, …)
```

Trigger “t3” applies to all links in module “m1”.

```dxl
trigger ("t3", module->"m1"->links, post, create, 10, …)
```

Specific module triggers, including all object level and attribute level triggers, are stored in the module to which they apply. If the trigger specification does not name a specific module, they are stored in the current module.

For modules, you can also restrict to a particular type of module: formal, link, or descriptive.

If you want an object trigger to apply only to the current object at definition time, you must give its absolute number as a specific argument:

```dxl
project->module->object->"13"
```

To simplify the notation you can omit mention of the project or module levels when you want the current project or module. The example becomes:

```dxl
attribute->"Cost"
```

In summary, if you do not mention a level, you mean the current position in the Rational DOORS schema at the time of definition.
Trigger events

There are eight classes of event, with synonyms:

- **open, read**
  Synonyms for the same fundamental event, open is usually used with projects and modules, while read is used with objects.

- **close**
  Triggered when Rational DOORS is about to close a project or a module.

- **write, save, modify**
  Synonyms for wanting to make a change; currently only supported for attribute modification.

- **sync**
  Triggered when the current object changes in a formal module.

- **drag**
  Triggered when the user starts a drag operation from a formal module object.

- **drop**
  Triggered when data is dragged from another application and dropped onto a displayed formal module object.

- **create**
  Applies to discussions, comments and links, fires on their creation.

- **delete**
  Applies to discussions, comments and links, fires on their deletion.

If you try to define an unsupported trigger combination, an error message is issued.

Trigger priority

Triggers are assigned an integer priority; lower valued priorities are executed before higher valued priorities.

Persistent versus dynamic triggers

There are two further classes of trigger:

- **persistent**
  Stored in Rational DOORS; once defined, persists between sessions until deleted.

- **dynamic**
  Not stored; persists only for the loaded lifetime of the project or module that defines it.

**Note:** In Rational DOORS 8.2 and later versions, drag-and-drop triggers can only be dynamic.
Trigger constants

This section lists constants that are used in the definition of triggers. Some are defined through internal data types; others are of type TriggerStatus.

levels

A level can be one of the following values:
- project
- module
- object
- attribute
- links
- discussion
- comment

level modifiers

A level modifier can be one of the following values:
- all
- formal
- link
- descriptive

These values specify the type of module affected.

event types

An event type can be one of the following values:
- pre
- post

event names

An event name can be one of the following values:
- open
- read
Trigger definition

This section defines an operator for assembling triggers and functions for triggers. They use internal data types or the data type Trigger.

Trigger level assembly

The -> operator is used to describe the extent to which a trigger is applied:

Syntax

The syntax for using the -> operator is as follows:

- `l -> l2`
- `l -> mod`
- `l -> string name -> mod`
- `l -> string name -> string name2`

where:

- `l l2` are levels: project, module, object or attribute
- `name name2` are strings
- `mod` is a modifier: all, formal, link, or descriptive

Operation

The operator combines trigger level descriptions and specifies the scope of a trigger.

Example

- This attribute-level trigger is applied to the Cost attribute in the module named URD in the current folder:
  
  module->"URD"->attribute->"Cost"

- This module-level trigger is applied to all formal modules in the current project:
  
  module->all->formal
These level descriptors are invalid if there is no current project.

- This module-level trigger is applied to all formal modules in the database:
  \texttt{project->all->module->formal->all}

- This object-level trigger is applied to all formal modules in the \texttt{improvements} project:
  \texttt{project->"improvements"->module->all->formal->object->all}

- This object-level trigger is applied to the current module:
  \texttt{module->object->"23"}

  This level descriptor is invalid if there is no current module.

\section*{trigger(persistent)}

\textbf{Declaration}

\begin{verbatim}
Trigger trigger(string name,
  l,
  t,
  e,
  int p,
  string dxl)
\end{verbatim}

where:

- \( l \) is a level: \texttt{project,module,object,attribute,attrdef,attrtype,or\ links}
- \( t \) is a type: \texttt{pre\ or\ post}
- \( e \) is an event: \texttt{open,read,close,save,modify,\ sync,create,or\ delete}
- \( p \) is a priority: indicates the order that similar triggers are executed
- \( dxl \) the dxl code that will be executed by the trigger

\textbf{Operation}

Creates a trigger, named \texttt{name}, at level \( l \), of type \( t \), of event \( e \), with priority \( p \), and code \texttt{dxl}. If the operation fails, the function returns \texttt{null}. If the user does not have the appropriate modify access, the call fails.

- To create a stored database wide trigger, the user must have modify access to the database root folder.
- To create a trigger stored in a project, the user must have modify access to the project.
- To create a trigger stored in a module, the user must have modify access to the module.

Optionally, the level can be a compound level description.
These triggers are persistent between sessions, and so need be created only once.

Example

This example creates a project level, pre-type, open event trigger of priority 10, using a program stored in $DOORSHOME/lib/dxl/triggers/projOpen.dxl:

```dxl
Trigger t1 = trigger("init", project,
    pre, open, 10,
    "#include <triggers/projOpen>")
```

This example sets up a trigger, which is executed when any module is about to be closed:

```dxl
Trigger t2 = trigger("mod", module->all,
    pre, close, 10,
    "#include <triggers/modClose>")
```

## trigger(dynamic)

### Declaration

```dxl
Trigger trigger(l, 
    e, 
    int p,
    {bool pre(Trigger)|
    void post(Trigger)})
```

where:

- `l` is a level: project, module, object, attribute, attrdef, attrtype, or links
- `e` is an event: open, read, close, save, modify, sync, create or delete
- `p` is a priority: indicates the order that similar triggers are executed

### Operation

Creates a dynamic trigger, which is not persistent between sessions, at level `l`, of event `e` and priority `p`. The `pre` callback function determines whether the operation happens or not. The callback function for a `post` event is a `void` function.

Optionally, the level can be a compound level description.
delete(trigger)

Declaration

string delete(string name,
   l,
   [string name2,]
   t,
   e,
   int p,)

string delete(Trigger &d)

where:

l is a level: project, module, object, attribute, attrdef, attrtype, or links

t is a type: pre or post

e is an event: open, read, close, save, modify, sync, create or delete

p is a priority: indicates the order that similar triggers are executed

Operation

The first form deletes the specified trigger. The second form deletes trigger d, and sets d to null. If the operation succeeds, returns null; otherwise, returns an error message. If the user does not have the appropriate modify access, the call fails.

To delete a stored database wide trigger, the user must have modify access to the database root folder. To delete a trigger stored in a project, the user must have modify access to the project. To delete a trigger stored in a module, the user must have modify access to the module.

This example deletes all triggers:

Trigger t

for t in current Project do delete t

Trigger manipulation

This section defines functions that return information about, or modify triggers.
for trigger in database

Syntax

for t in database do {
...
}

where:

\[t\] is a variable of type Trigger

Operation

Assigns trigger \[t\] to be each successive database wide trigger.

for trigger in project

Syntax

for t in project do {
...
}

where:

\[t\] is a variable of type Trigger

\[project\] is a variable of type Project

Operation

Assigns trigger \[t\] to be each successive trigger in the specified project, and in any open modules in the project. The appropriate modules in the project must be open to allow access to the relevant trigger information. It includes all subprojects.

Example

This example deletes all triggers:

Trigger t

for t in current Project do delete t
for trigger in module

Syntax

for t in m do {
    ...  
}

where:

m is a variable of type Module

t is a variable of type Trigger

Operation

Assigns trigger t to be each successive trigger in m, which must be an open module to allow access to the relevant trigger information.

level, type, event(trigger)

These functions are used as shown in the following syntax:

level(Trigger t)
type(Trigger t)
event(Trigger t)

Operation

These functions return values for the level, type and event of trigger t, as follows:

<table>
<thead>
<tr>
<th>level</th>
<th>project module object attribute attrdef attrtype links</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>pre post</td>
</tr>
<tr>
<td>event</td>
<td>open read close save modify sync create delete</td>
</tr>
</tbody>
</table>

stringOf(trigger)

These functions are used as shown in the following syntax:

string stringOf(level)
string stringOf(type)
string stringOf(event)
Operation
Return the string version of trigger level *level*, the trigger type *type*, or trigger event *event*, as follows:

* level project module object attribute
  attrdef attrtype links

* type pre post

* event open read close save modify sync

**attribute(trigger)**

**Declaration**

```plaintext
string attribute(Trigger t)
```

**Operation**

Returns the name of the attribute to which trigger *t* applies, (if there is one); otherwise, returns *null*.

**attrdef(trigger)**

**Declaration**

```plaintext
AttrDef attrdef(Trigger t)
```

**Operation**

Returns the name of the attribute about to be saved for attribute pre-save triggers. For pre-open attribute triggers, returns *null*.

**current(trigger)**

**Declaration**

```plaintext
Trigger current()
```

**Operation**

Gets the current trigger handle in persistent trigger code.

**dxl(trigger)**

**Declaration**

```plaintext
string dxl(Trigger t)
```
Operation
Returns the DXL code associated with trigger t.

kind

Declaration
string kind(Trigger t)
Operation
Returns the kind of trigger t: one of dynamic, stored or builtin.

levelModifier

Declaration
string levelModifier(Trigger t)
Operation
Returns the module level modifier of trigger t, which can be one of the following values:

 "F" formal module
 "L" link module
 "D" descriptive module

name(trigger)

Declaration
string name(Trigger t)
Operation
Returns the name of trigger t.

object(trigger)

Declaration
string object(Trigger t)
Object object(Trigger t)
module

Declaration

Module module(Trigger t)
string module(Trigger t)
Module module(Trigger t, int unused)

Operation

Fetches the module associated with the specified trigger.

The notion of associated module is as follows:

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Returns...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-open module and post-close module triggers</td>
<td>Normally a NULL module is returned. When a non-null module is returned, it will be the module against which this trigger last fired.</td>
</tr>
<tr>
<td>Post-open module trigger</td>
<td>The current version module or null is returned. When a baseline is opened, this perm will return the current version of the module only when that current version has been separately loaded.</td>
</tr>
<tr>
<td>Pre-close module trigger</td>
<td>The current version module or null is returned. When a baseline is closed, this perm will return the current version of the module only when that current version has been separately loaded.</td>
</tr>
<tr>
<td>Data-related triggers within a module (object and attribute open/sync, and so on)</td>
<td>The tree where the data resides.</td>
</tr>
</tbody>
</table>

Note: Data with a baseline will return the baseline module.

For the following triggers, the associated module is the module containing the data (this will be a baseline when the data is in a baseline):

- object attribute, pre-save
- object attribute, post-save
- link attribute, pre-save
- link attribute, post-save

The third form is as Module module(Trigger), but this variant will return baselined modules when a module-level trigger is running against a baseline of a module. For non-module triggers, the returned module is the same as Module module(Trigger).

The unused integer parameter should be 0.
**version**

**Declaration**

ModuleVersion version(Trigger t)

**Operation**

Returns the version information pertaining to the specified trigger. The returned value will be null in the case that version information is not appropriate to the trigger.

It is not currently possible to associate a trigger with a specific module version, and thus only executing triggers have an associated version.

**link**

**Declaration**

Link link(Trigger t)

**Operation**

When a trigger fires because of an operation on a link, for example modification of an attribute) this perm provides access to the corresponding link. In all other cases null is returned.

**value**

**Declaration**

void value(Trigger t, Buffer b)

**Operation**

Similar to string value(Trigger), but returns in buffer the RTF, inclusive of any OLE objects, of the new value (where that is appropriate).

The creation and deletion of b is the responsibility of the user.

**priority**

**Declaration**

int priority(Trigger t)

**Operation**

Returns the priority of trigger t. Lower numbers have higher priority.
**trigger status**

A trigger status can be one of the following values:

- `trigPreConPass`
- `trigPreConFail`
- `trigRunOK`
- `trigError`

These constants are of type `TriggerStatus`. They are used with the `set` function. They are assigned to persistent pre-event triggers to set a return condition.

**set(trigger status)**

**Declaration**

```c
void set(TriggerStatus ts)
```

**Operation**

Sets a return condition in the DXL code assigned to persistent pre-event triggers. Possible values are: `trigPreConPass`, `trigPreConFail`, `trigRunOK`, and `trigError`.

**Example**

```c
Trigger t1 = trigger("t1", module->object->"1", pre, modify, 10, "#include <trigger.dxl>")
```

In your `trigger.dxl` code, include one of the following set functions:

- `set(trigPreConPass)` - This causes the event that is associated with the trigger to pass always.
- `set(trigPreConFail)` - This causes the event that is associated with the trigger to fail always.

If the `trigPreConFail` value is used, then the object that is associated with the trigger event cannot be modified.

**stored**

**Declaration**

```c
string stored(Trigger t)
```

**Operation**

Returns the name of the module where trigger `t` is stored.
scope

Declaration

string scope(Trigger t)

Item scope(Trigger t)

Operation

Returns the item (or its unqualified name) to which the specified trigger applies. If the item is a project, then the trigger applies to all modules within the project. For static triggers, this returns the same as the stored () perm.

value

Declaration

string value(Trigger t)

Operation

Similar to string value(Trigger), but returns the value being proposed for attribute modification by trigger t.

Triggers review

The following tables show what information is available to triggers of various types.
Dynamic triggers

<table>
<thead>
<tr>
<th></th>
<th>kind</th>
<th>dynamic</th>
<th>dynamic</th>
<th>dynamic</th>
<th>dynamic</th>
<th>dynamic</th>
<th>dynamic</th>
<th>dynamic</th>
<th>dynamic</th>
<th>dynamic</th>
<th>dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>module</td>
<td>module</td>
<td>object</td>
<td>object</td>
<td>attribute</td>
<td>module</td>
<td>module</td>
<td>object</td>
<td>object</td>
<td>attribute</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>pre</td>
<td>pre</td>
<td>pre</td>
<td>pre</td>
<td>pre</td>
<td>post</td>
<td>post</td>
<td>post</td>
<td>post</td>
<td>post</td>
<td></td>
</tr>
<tr>
<td>event</td>
<td>open</td>
<td>close</td>
<td>open</td>
<td>sync</td>
<td>save</td>
<td>open</td>
<td>close</td>
<td>open</td>
<td>sync</td>
<td>save</td>
<td></td>
</tr>
<tr>
<td>priority</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>attribute</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes¹</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes¹</td>
</tr>
<tr>
<td>levelModifier</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>name</td>
<td>yes²</td>
<td>yes²</td>
<td>yes²</td>
<td>yes²</td>
<td>yes²</td>
<td>yes²</td>
<td>yes²</td>
<td>yes²</td>
<td>yes²</td>
<td>yes²</td>
<td>yes²</td>
</tr>
<tr>
<td>string object</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>string module</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>value</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>dxl</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Module module</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Object object</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>attrdef</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

¹ Only if trigger is on a named attribute.

² Trigger name generated by system
Persistent triggers

<table>
<thead>
<tr>
<th>kind</th>
<th>stored</th>
<th>stored</th>
<th>stored</th>
<th>stored</th>
<th>stored</th>
<th>stored</th>
<th>stored</th>
<th>stored</th>
<th>stored</th>
<th>stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>module</td>
<td>module</td>
<td>object</td>
<td>object</td>
<td>attribute</td>
<td>module</td>
<td>module</td>
<td>object</td>
<td>object</td>
<td>attribute</td>
</tr>
<tr>
<td>type</td>
<td>pre</td>
<td>pre</td>
<td>pre</td>
<td>pre</td>
<td>pre</td>
<td>post</td>
<td>post</td>
<td>post</td>
<td>post</td>
<td></td>
</tr>
<tr>
<td>event</td>
<td>open</td>
<td>close</td>
<td>open</td>
<td>sync</td>
<td>save</td>
<td>open</td>
<td>close</td>
<td>open</td>
<td>sync</td>
<td>save</td>
</tr>
<tr>
<td>priority</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>attribute</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes³</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes³</td>
</tr>
<tr>
<td>levelModifier</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>name</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>string object</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>string module</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>dxl module</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module object</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object object</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>attrdef</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

³ Only if trigger is on a named attribute

Drag-and-drop trigger functions

This section defines functions that are used to setup drag or drop trigger callback functions, as well as those functions which can be used within them.

createDropCallback

Declaration

```c
void createDropCallback(int fmt, int type, void cb(Trigger), Trigger t)
```
Operation
When used in a callback for drag trigger \( t \), this registers a DXL callback functions \( cb \) to be run when the drop target application requests data in the specified clipboard format \( fmt \) and media \( type \) with a value included in the bitmap value \( type \). The \( fmt \) and \( type \) argument should match the format of data which is supplied by the callback function using \( \text{setDropString()}, \text{setDropList()} \) etc.

registeredFormat

Declaration

\[
\text{int registeredFormat(string formatName)}
\]

Operation

Returns the format ID for the specified format name. If the named format has not already been registered, then this perm registers it.

dropDataAvailable

Declaration

\[
\text{bool dropDataAvailable(format, int type, Trigger t)}
\]

Operation

Returns true if dragged data is available in the specified clipboard format, which may be specified as a string registered format name, or a format ID number. The \( type \) argument is used to specify which media formats should be checked for.

droppedString

Declaration

\[
\text{string droppedString(format, Trigger t[, bool unicode])}
\]

Operation

When used in a callback function for a drop event trigger \( t \), this returns any text supplied in the specified clipboard format by the data source application. The \( format \) argument can be either the name of a registered clipboard format (a string), or a format ID (int). If the \( unicode \) argument is specified and is true, and the clipboard format is a registered (non-standard) clipboard format, then the string data in the clipboard will be assumed to be in wide-char Unicode format.

droppedAttrTextAvailable

Declaration

\[
\text{bool droppedAttrTextAvailable(string attr, Trigger t)}
\]
Operation
When used in a callback function for a drop event trigger \( t \), this function tells whether a dragged text value is available for the named Rational DOORS Object attribute \( \textit{attr} \) from the drag source object. Returns true when the drag source is another Rational DOORS client, and the \( \textit{attr} \) is an Object attribute in the drag source module whose value can be expressed as a string, and to which the current user in the source has read access.

\[\text{droppedAttributeText} \]

Declaration

\[
\text{string droppedAttributeText(string \textit{attr}, Trigger \textit{t})}
\]

Operation
When used in a callback function for a drop event trigger \( t \), if the drag source is a Rational DOORS client this returns the text form of the named Object attribute \( \textit{attr} \). Returns an empty (null) string when there is no accessible text value corresponding to the named attribute.

\[\text{droppedAttrRichTextAvailable} \]

Declaration

\[
\text{bool droppedAttrRichTextAvailable(string \textit{attr}, Trigger \textit{t})}
\]

Operation
When used in a callback function for a drop event trigger \( t \), this tells whether a dragged Rich Text value (excluding OLE objects) is available for the named Rational DOORS Object attribute \( \textit{attr} \) from the drag source object. This returns true when the drag source is another Rational DOORS client, and the named attribute is an Object attribute with base-type Text or String in the drag source module, and to which the current user in the source has read access.

\[\text{droppedAttributeRichText} \]

Declaration

\[
\text{string droppedAttributeRichText(string \textit{attr}, Trigger \textit{t})}
\]

Operation
When used in a callback function for a drop event trigger \( t \), this returns the Rich Text value (excluding OLE objects) of the named Object attribute \( \textit{attr} \), when the drag source is a Rational DOORS client. This returns an empty (null) string when the named attribute is not of base type String or Text.

\[\text{droppedAttrOLETextAvailable} \]

Declaration

\[
\text{bool droppedAttrOLETextAvailable(string \textit{attr}, Trigger \textit{t})}
\]
Operation
When used in a callback function for a drop event trigger \( t \), this tells whether a dragged Rich Text value (including OLE objects) is available for the named Rational DOORS Object attribute \( \text{attr} \) from the drag source object. This returns true when the drag source is another Rational DOORS client, and the named attribute is an Object attribute with base-type Text in the drag source module, and to which the current user in the source has read access.

droppedAttributeOLEText

Declaration

\[
\text{string droppedAttributeOLEText(string } \text{attr}, \text{ Trigger } t)\
\]

Operation
When used in a callback function for a drop event trigger \( t \), this returns the Rich Text form (including OLE objects) of the named Object attribute \( \text{attr} \), when the drag source is a Rational DOORS client. This returns an empty (null) string when the named attribute is not of base type Text.

draggedObjects

Declaration

\[
\text{Skip draggedObjects()}\
\]

Operation
This returns a Skip list of the objects in the selection where the latest drag has begun. Its return value is only valid within the context of a drag trigger or a drop callback registered by a drag trigger.

droppedList

Declaration

\[
\text{Skip droppedList(format, Trigger } t)\
\]

Operation
When used in a callback function for a drop event trigger \( t \), this returns any list of strings supplied in the specified clipboard format by the data source application. The \( \text{format} \) argument can be either the name of a registered clipboard format (a string), or a format ID (int).

The data should be supplied as in the standard CF_HDROP format.

setDropString

Declaration

\[
\text{string setDropString(int } \text{fmt, Trigger } t, \text{ string } s [, \text{ bool unicode}])\
\]
Operation
When used in a callback for a drag trigger \( t \), or in a drop callback function registered by \( \text{createDropCallback}() \),
this passes the string \( s \) to the drop target in the specified clipboard format \( fmt \), in \( \text{TYMED_HGLOBAL} \) media type.
If \( fmt \) is a non-standard registered clipboard format and \( \text{unicode} \) is specified and is true, then the string data will be supplied in wide-char Unicode format.
Returns null on success, and an error string on failure.

### setDropList

**Declaration**

```dxl
definition string setDropList(int fmt, Trigger t, Skip sk)
```

**Operation**
When used in a callback for a drag trigger \( t \), or in a drop callback function registered by \( \text{createDropCallback}() \),
this passes the strings in the supplied Skip list to the drop target in the specified clipboard format \( fmt \), in \( \text{TYMED_HGLOBAL} \) media type, as supplied in the standard CF_HDROP clipboard format.
Returns null on success, and an error string on failure.

### insertDroppedPicture

**Declaration**

```dxl
definition bool insertDroppedPicture(Object, Trigger t, int fmt[, int type])
```

**Operation**
When used in a callback for a drop trigger \( t \), and when the specified Object is an editable Picture object, and if picture data is available in the specified format \( fmt \) and \( type \), then this replaces the Object's picture with the picture from the drag source.
If \( type \) is \( \text{TYMED_MFPICT} \) or \( fmt \) is \( \text{CF_METAFILEPICT} \), then Windows Metafile data will be expected. Otherwise, if \( fmt \) is \( \text{CF_BITMAP} \) then a Device Dependent Bitmap is expected. Otherwise, a Device Independent Bitmap is expected.
The default value for \( type \) is \( \text{TYMED_MFPICT} \) for \( \text{CF_METAFILEPICT} \) clipboard format, and \( \text{TYMED_GDI} \) for \( \text{CF_BITMAP} \), \text{CF_DIB} and all other formats.
Returns true on success, false on failure.

### saveDroppedPicture

**Declaration**

```dxl
definition bool saveDroppedPicture(Trigger t, string filename, int fmt[, int type])
```
Operation

When used in a callback for a drop trigger t, this saves any picture data available in the specified format fmt and data type type in the file specified by the full path filename.

If type is TYMED_MFPICT or fmt is CF_METAFILEPICT then Windows Metafile data will be expected. Otherwise, if fmt is CF_BITMAP then a Device Dependent Bitmap is expected. Otherwise, a Device Independent Bitmap is expected.

The default value for type is TYMED_MFPICT for CF_METAFILEPICT clipboard format, and TYMED_GDI for CF_BITMAP, CF_DIB and all other formats.

Returns true on success, false on failure.

Example

The following two examples, when run in the global context, define drag-and-drop triggers that give some control over the dragging and dropping of data to and from Rational DOORS clients.

Drag trigger example:

```dxl
/*
dragTrigger.inc
*/

// Drop callback to supply Object Text in CF_OEMTEXT format.
void dropCB(Trigger t)
{
    Object o = object(t)
    setDropString(CF_OEMTEXT, t, o."Object Text" "")
}

string formatName = "RichEdit Text and Objects"

// Test drop callback
void testCB(Trigger t)
{
    Object o = object(t)
    setDropString(registeredFormat(formatName), t, o."Object Text" "")
}

// Drag trigger: Register callbacks to set CF_OEMTEXT and CF_HDROP
// format data.
bool preDrag(Trigger t)
{  
    Object o = object(t)  
    createDropCallback(CF_OEMTEXT, TYMED_HGLOBAL, dropCB, t)  
    createDropCallback(registeredFormat(formatName), TYMED_HGLOBAL, testCB, t)  
    return true  
}

trigger(project->all->module->all->object->all,drag,1,preDrag)

Drop trigger example:

/*
   dropTrigger.inc
   */
// Append registered format drag-drop data info to the buffer for display.
void appendData(Buffer &b, string fmtName, Trigger t, bool unicode)
{
    int tymed
    int types = 0

    for (tymed = TYMED_HGLOBAL; tymed <= TYMED_ENHMF; tymed *= 2)
    {
        if (dropDataAvailable(fmtName,tymed,t))
        {
            types |= tymed
        }
    }
    if (types > 0)
    {
        int fmt = registeredFormat(fmtName)
        b += fmt " (" fmtName ", " types ") :\n"
        b += " " droppedString(fmtName,t,unicode) "\n"
    }
void appendText(Buffer &b, string attrName, bool isSpecial, Trigger t) {
    if (droppedAttrTextAvailable(attrName, t, isSpecial)) {
        b += "Attribute Text: " attrName "::\n"
        b += "   " droppedAttributeText(attrName, t, isSpecial) "\n"
    }
}

void appendRTF(Buffer &b, string attrName, Trigger t) {
    if (droppedAttrRichTextAvailable(attrName, t)) {
        b += "Attribute RichText: " attrName "::\n"
        b += "   " droppedAttributeRichText(attrName, t) "\n"
    }
}

void appendOLE(Buffer &b, string attrName, Trigger t) {
    if (droppedAttrOLETextAvailable(attrName, t)) {
        b += "Attribute OLE Text: " attrName "::\n"
        b += "   " droppedAttributeOLEText(attrName, t) "\n"
    }
}

// Custom trigger: Displays a dialog listing available clipboard formats
// from drag and drop, and displays any string data and list data.
// Prompts the user to insert any available picture-format data if the
// module is open for edit.
bool preDrop(Trigger t)
if (!confirm("Run custom trigger?"))
{
    return true
}
Buffer b = create
Object o = object(t)
int fmt
    int types
    // Check for available data in standard clipboard formats.
for (fmt = 1; fmt < CF_MAX; fmt++)
{
    int tymed = TYMED_HGLOBAL
    types = 0
    for (tymed = TYMED_HGLOBAL; tymed <= TYMED_ENHMF; tymed *= 2)
    {
        if (dropDataAvailable(fmt,tymed,t))
        {
            types |= tymed
        }
    }
    if (types > 0)
    {
        b += fmt " (" clipboardFormatName(fmt) ", " types ") :
";
        if (fmt == CF_HDROP)
        {
            Skip skp = droppedList(fmt,t)
            string s
            for s in skp do
            {
                b += " - " s "\n"
            }
            delete skp
        }
    }
else if (fmt == CF_DIB || fmt == CF_BITMAP || fmt == CF_METAFILEPICT)
{
    if (isEdit(module o) && confirm("Insert picture format "
    clipboardFormatName(fmt) "?"))
    {
        if (formatUNKNOWN != getPictFormat(o))
        {
            // Dropping onto a picture object -
            // replace the existing picture
            insertDroppedPicture(o,t,fmt)
            refresh(module o)
        }
        else
        {
            // Not a picture object: append a new one.
            string filename = tempFileName()
            int tymed = TYMED_GDI
            int picFmt = formatBMP
            if (fmt == CF_METAFILEPICT)
            {
                tymed = TYMED_MFPICT
                picFmt = formatWMF
            }
            if (saveDroppedPicture(t, filename, fmt, tymed))
            {
                insertPictureFileAfter(filename, picFmt, o)
                deleteFile(filename)
                refresh(module o)
            }
        }
    }
}
b += "  " droppedString(fmt,t) "\n"
}
}
}

if (droppedAttrTextAvailable("Object Heading",t))
{
    if (confirm("Replace Object Heading?"))
    {
        o."Object Heading" = droppedAttributeText("Object Heading",t)
    }
}

if (droppedAttrRichTextAvailable("Object Text",t))
{
    if (confirm("Replace Object Rich Text?"))
    {
        o."Object Text" = droppedAttributeRichText("Object Text",t)
    }
}

if (droppedAttrOLETextAvailable("Object Text",t))
{
    if (confirm("Replace Object Rich Text with OLE?"))
    {
        o."Object Text" = droppedAttributeOLEText("Object Text",t)
    }
}

// Check for specific registered clipboard formats.
appendData(b,"DOORS Object URL",t,false)
appendData(b,"RichEdit Text and Objects",t,false)
appendText(b,"Object Heading",false,t)
appendText(b,"Object Text",false,t)
appendText(b,"Last Modified Time",true,t)
appendData(b,"UniformResourceLocator",t,false)
b += "\nonto Object " o."Absolute Number" "

// Display the results.
DB thedb = create (module(o), "Dropped data")
DBE thetext = text(thedb,"",stringOf(b),200,true)
block thedb
  destroy thedb
  delete b
  return false
}

trigger(project->all->module->all->object->all,drop,1,preDrop)
This chapter describes the page setup functions.

- Page attributes status
- Page dimensions
- Document attributes
- Page setup information
- Page setup management

## Page attributes status

This section describes the page setup functions that return the status of a page attribute or set it. They are intended for use in exporters.

In each case there are two versions of the function that gets the status of a page attribute: one for a specific page; the other with no page specified, which operates on the current page. Similarly, there are two versions of each function that sets the status of a page attribute. The functions that get or set data for a specific page use the data type `PageLayout`.

### Get page properties status

#### Declaration

```plaintext
bool pageChangeBars([PageLayout myPageSetup])
bool pagePortrait([PageLayout myPageSetup])
bool pageRepeatTitles([PageLayout myPageSetup])
bool pageTitlePage([PageLayout myPageSetup])
```

where:

- `myPageSetup` Specifies a page setup

#### Operation

Returns `true` for the properties described below on `myPageSetup`, or if `myPageSetup` is omitted, on the current page; otherwise, it returns `false`.

- `pageChangeBars` Shows change bars
- `pagePortrait` Is portrait
pageRepeatTitles  Repeats titles on every page  
pageTitlePage  Shows a title page  

Set page properties status

Declaration

bool pageChangeBars([PageLayout myPageSetup,]  
  bool expression)

bool pagePortrait([PageLayout myPageSetup,]  
  bool expression)

bool pageRepeatTitles([PageLayout myPageSetup,]  
  bool expression)

bool pageTitlePage([PageLayout myPageSetup,]  
  bool expression)

where:

myPageSetup  Specifies a page setup  
expression  Is an expression

Operation

Sets the properties described below on myPageSetup, or if myPageSetup is omitted, on the current page. Returns true if the operation succeeds; otherwise, returns false.

<table>
<thead>
<tr>
<th>Argument expression</th>
<th>Evaluates true</th>
<th>Evaluates false</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageChangeBars</td>
<td>Shows change bars</td>
<td>Hides change bars</td>
</tr>
<tr>
<td>pagePortrait</td>
<td>Sets portrait</td>
<td>Sets landscape</td>
</tr>
<tr>
<td>pageRepeatTitles</td>
<td>Repeats titles on every page</td>
<td>Shows titles on first page only</td>
</tr>
<tr>
<td>pageTitlePage</td>
<td>Shows a title page</td>
<td>Suppresses a title page</td>
</tr>
</tbody>
</table>

Page dimensions

This section describes the page setup functions that return or set the size of a page dimension.

In each case there are two versions of the function that gets the size of a page dimension: one for a specific page; the other with no page specified, which operates on the current page. Similarly, there are two versions of each function that sets the size of a page dimension. The functions that get or set dimensions for a specific page use the data type PageLayout.
Get page dimension

Declaration

int pageSize([PageLayout myPageSetup])
int pageWidth([PageLayout myPageSetup])
int pageHeight([PageLayout myPageSetup])
int pageTopMargin([PageLayout myPageSetup])
int pageBottomMargin([PageLayout myPageSetup])
int pageLeftMargin([PageLayout myPageSetup])
int pageRightMargin([PageLayout myPageSetup])

where:

myPageSetup Specifies a page setup

Operation

Returns the size as described below on myPageSetup, or if myPageSetup is omitted, on the current page.

pageSize Page size indicated by 0 (A4), 1 (A3), 2 (A5), 3 (legal), 4 (letter), 5 (custom)
pageWidth Page width in mm
pageHeight Page height in mm
pageTopMargin Top margin in mm
pageBottomMargin Bottom margin in mm
pageLeftMargin Left margin in mm
pageRightMargin Right margin in mm

Set page dimension

Declaration

bool pageSize([PageLayout myPageSetup, ]
            int dimension)
bool pageWidth([PageLayout myPageSetup, ]
               int dimension)
bool pageHeight([PageLayout myPageSetup, ]
                int dimension)
bool pageTopMargin([PageLayout myPageSetup, ]
    int dimension)
bool pageBottomMargin([PageLayout myPageSetup, ]
    int dimension)
bool pageLeftMargin([PageLayout myPageSetup, ]
    int dimension)
bool pageRightMargin([PageLayout myPageSetup, ]
    int dimension)

where:

    myPageSetup Specifies a page setup
    dimension Specifies a dimension

Operation
Sets the size of the dimension described below on myPageSetup, or if myPageSetup is omitted, on the current page. Returns true if the operation succeeds; otherwise, returns false.

    pageSize Page size indicated by 0 (A4), 1 (A3), 2 (A5), 3 (legal), 4 (letter), 5 (custom)
    pageHeight Page height in mm
    pageWidth Page width in mm
    pageTopMargin Top margin in mm
    pageBottomMargin Bottom margin in mm
    pageLeftMargin Left margin in mm
    pageRightMargin Right margin in mm

Example

const int paperA4 = 0,
    paperA3 = 1,
    paperA5 = 2,
    paperLegal = 3,
    paperLetter = 4,
    paperCustom = 5

if (pageSize == paperCustom) {
    // do something specific
}
Document attributes

This section describes the page setup functions that return or set a document attribute. These are features of a complete document rather than a page.

For pageBreakLevel, pageTOCLevel and pageHeaderFooter, there are two versions of the function that gets the document attribute: one for a specific page; the other with no page specified, which operates on the current page. Similarly, there are two versions of these functions that set the document attribute. The functions that get or set data for a specific page use the data type PageLayout. Note that a statement such as ‘pageBreakLevel = 1’ is not supported.

pageBreakLevel, pageTOCLevel(get)

Declaration

int pageBreakLevel([PageLayout myPageSetup])
int pageTOCLevel([PageLayout myPageSetup])

where:

myPageSetup Specifies a page setup

Operation

Returns the document attribute as described below on myPageSetup, or if myPageSetup is omitted, on the current page.

pageBreakLevel Heading level at which a page break is automatically inserted
pageTOCLevel Lowest heading level included in table of contents

pageBreakLevel, pageTOCLevel(set)

Declaration

bool pageBreakLevel([PageLayout myPageSetup, ]
int level)
bool pageTOCLevel([PageLayout myPageSetup, ]
int level)

where:

myPageSetup Specifies a page setup
level Specifies a level
Operation

Sets the document attribute described below on myPageSetup, or if myPageSetup is omitted, on the current page. Returns true if the operation succeeds; otherwise, returns false.

pageBreakLevel: Heading level at which a page break is automatically inserted
pageTOCLevel: Lowest heading level included in table of contents

pageHeaderFooter(get)

Declaration

string pageHeaderFooter([PageLayout myPageSetup, int fieldNumber])

where:

myPageSetup: Specifies a page setup
fieldNumber: Identifies a header or footer field

Operation

Returns the header or footer string defined for myPageSetup, or if myPageSetup is omitted, for the current page, as follows:

<table>
<thead>
<tr>
<th>fieldNumber for page type</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>left header</td>
</tr>
<tr>
<td>center header</td>
</tr>
<tr>
<td>right header</td>
</tr>
<tr>
<td>left footer</td>
</tr>
<tr>
<td>center footer</td>
</tr>
<tr>
<td>right footer</td>
</tr>
</tbody>
</table>

pageHeaderFooter(set)

Declaration

bool pageHeaderFooter([PageLayout myPageSetup,] int fieldNumber, string s)
where:

- `myPageSetup` Specifies a page setup
- `fieldNumber` Identifies a header or footer field
- `s` Is the string to be placed in the specified field

**Operation**

Places the header or footer string in the specified field (see the `pageHeaderFooter(get)` function) on `myPageSetup`, or if `myPageSetup` is omitted, on the current page. Returns `true` if the operation succeeds; otherwise, returns `false`.

---

**pageExpandHF**

**Declaration**

```dxl
string pageExpandHF(string HF,
                   string thisPage,
                   string maxPage)
```

**Operation**

Takes a header or footer string, `HF`, a current page number as a string, and a maximum page number as a string, and returns the string to be printed. Page numbers are passed as strings to permit roman and other numerals. Typically, the `HF` value is returned from the `pageHeaderFooter(get)` function.

The options are:

- `&N` Current page number, for a contents page in Roman numerals; not available on a title page
- `&C` Total page count; not available for title page or contents pages
- `&M` Current module name
- `&P` Project name
- `&V` Current version of module
- `&U` User name
- `&D` Session date
- `&T` Time of printing
- `&A` Rational DOORS product name
- `&B` Rational DOORS product version
Example

This example prints Page 1 of 10:

\[
\text{print pageExpandHF("Page &N of &C", "1", "10")}
\]

Page setup information

This section describes the page setup functions that return or set specific information.

For pageColumns, and pageFormat, there are two versions of the function that gets the layout information: one for a specific page; the other with no page specified, which operates on the current page. Similarly, there are two versions of each function that sets the layout information. The functions that get or set layout information for a specific page use the data type PageLayout.

Setting current page setup

The assignment operator = can be used as shown in the following syntax:

\[
\text{current = PageLayout setup}
\]

Makes setup the current page setup, provided the user has read access to the page setup. See also, the current (page setup) function.

For large DXL programs, when you set the current page setup, cast the current on the left hand side of the assignment to the correct type. This speeds up the parsing of the DXL program, so when your program is first run, it is loaded into memory quicker. It does not affect the subsequent execution of your program. So:

\[
\text{current = newCurrentPageSetup}
\]

becomes

\[
(\text{current ModuleRef__}) = \text{newCurrentPageSetup}
\]

Note that this cast only works for assignments to current. It is not useful for comparisons or getting the value of the current page setup.

current(page setup)

Declaration

PageLayout current()

Operation

Returns the current page setup.
pageColumns, pageFormat(get)

**Declaration**

```c
int pageColumns([PageLayout myPageSetup])
int pageFormat([PageLayout myPageSetup])
```

where:

- `myPageSetup`: Specifies a page setup

**Operation**

Returns the information described below on `myPageSetup`, or if `myPageSetup` is omitted, on the current page.

- `pageColumns`: Column style indicated by 0 (filled), 1 (table), 2 (not marked)
- `pageFormat`: Page format indicated by 0 (columns), 1 (book)

pageColumns, pageFormat(set)

**Declaration**

```c
bool pageColumns([PageLayout myPageSetup,] int style)
bool pageFormat([PageLayout myPageSetup,] int style)
```

where:

- `myPageSetup`: Specifies a page setup
- `style`: Specifies a style

**Operation**

Sets the information described below on `myPageSetup`, or if `myPageSetup` is omitted, on the current page. Returns `true` if the operation succeeds; otherwise, returns `false`.

- `pageColumns`: Column style indicated by 0 (filled), 1 (table), 2 (not marked)
- `pageFormat`: Page format indicated by 0 (columns), 1 (book)
pageTitle

**Declaration**

```
bool pageTitlePage()
bool pageTitlePage(PageLayout)
```

**Operation**

These functions allow the user to get the signature page setting for either the current page layout or the specified one.

pageSignaturePage

**Declaration**

```
bool pageSignaturePage(bool)
bool pageSignaturePage(PageLayout, bool)
```

**Operation**

These functions allow the user to set the signature page setting for either the current page layout or the specified one.

pageIncludeFilters

**Declaration**

```
bool pageIncludeFilters([PageLayout][, bool])
```

**Operation**

These functions allow the user to either set the **Include filter criteria on title page** setting, or, if a boolean parameter is not supplied, obtain the current setting.

If a PageLayout is not supplied, the operation will be performed on the current PageLayout.

pageIncludeSort

**Declaration**

```
bool pageIncludeSort([PageLayout][, bool])
```

**Operation**

These functions allow the user to either set the **Include sort criteria on title page** setting, or, if a boolean parameter is not supplied, obtain the current setting.

If a PageLayout is not supplied, the operation will be performed on the current PageLayout.
Page setup management

This section defines the functions that allow you to manage page setups.

create

Declaration
PageLayout create(string myPageSetup)

Operation
Creates the page setup myPageSetup.

delete

Declaration
bool delete(PageLayout myPageSetup)

Operation
Deletes the page setup myPageSetup. Returns true if the operation succeeds; otherwise, returns false.

isValidName

See “isValidName,” on page 296.

pageLayout

Declaration
PageLayout pageLayout(string myPageSetup)

Operation
Returns the page setup of myPageSetup.

pageName

Declaration
string pageName([PageLayout myPageSetup])

Operation
Returns the name of the page setup myPageSetup, or if myPageSetup is omitted, of the current page.
save(page setup)

Declaration

    bool save(PageLayout myPageSetup)

Operation

Saves the page setup myPageSetup. Returns true if the operation succeeds; otherwise, returns false.

for setup name in setups

Syntax

    for setupName in pageSetups database do {
        ...  
    }

where:

    setupName is a string variable

Operation

Assigns the string setupName to be each successive page setup name found in the database.

Example

    string setupName
    for setupName in pageSetups database do {
        print setupName "\n"
    }
Chapter 31
Tables

This chapter describes the table handling functions, many of which are useful for making exporters.

- Table concept
- Table constants
- Table management
- Table manipulation
- Table attributes

Table concept

In Rational DOORS, a table is an object hierarchy displayed in the form of a table.

The table’s top level is referred to as the table header object; for each row it has a sub-object, called a row object. These row objects, in turn, have sub-objects, which are the table cells.

Table constants

You can use the column alignment constants of type Justification for tables. For further information, see “Column alignment constants,” on page 688.

You define table borders using constants of type TableBorderStyle and TableBorderPosition.

Declaration

```dxl
const TableBorderStyle noborder
cost TableBorderStyle solidBorder
cost TableBorderStyle dottedBorder
cost TableBorderPosition left
cost TableBorderPosition right
cost TableBorderPosition top
cost TableBorderPosition bottom
```

Operation

These constants are used to define tables with the setCellBorder and setAllCellsBorder functions.
Table management

This section defines the table management functions.

table(create)

Declaration

Object table(Module m,
    int rows,
    int cols)

Object table(Object o,
    int rows,
    int cols)

Object table(before(Object o),
    int rows,
    int cols)

Object table(below(Object o),
    int rows,
    int cols)

Object table(last(below(Object o)),
    int rows,
    int cols)

Operation

The first form creates a table of size \textit{rows}, \textit{cols} as the first object in a module.
The second form creates a table of size \textit{rows}, \textit{cols} at the same level and immediately after object \textit{o}.
The third form creates a table of size \textit{rows}, \textit{cols} at the same level and immediately before the object \textit{o}.
The fourth form creates a table of size \textit{rows}, \textit{cols} as the first child of the object \textit{o}.
The fifth form creates a table of size \textit{rows}, \textit{cols} as the last child of the object \textit{o}.

Example

// create as first object
Object params = table(current Module, 10, 3)

// create at same level and after object
Object analysis = table(current object, 4, 4)

// create at same level and before object
Object revisions = table(before first current, noOfChanges, 3)
// create as first child
Object wordCount = table(below checkedObject, noOfWords, 2)

table

Declaration
bool table(Object o)

Operation
Returns true if o is a table header object; otherwise, returns false.
Use this function in an exporter that does not handle tables.

row

Declaration
bool row(Object o)

Operation
Returns true if o is a row header object; otherwise, returns false.

cell

Declaration
bool cell(Object o)

Operation
Returns true if o is a table cell; otherwise, returns false.

tableContents(get)

Declaration
bool tableContents(Module m)

Operation
Gets the status of tables in the specified module. It returns true for tables shown or false for tables hidden.
Example

```dxl
if (tableContents current Module &&
    table current Object) {
    dumpTable(current, outStream)
}
```

tableContents(set)

**Declaration**

```dxl
void tableContents(bool expression)
```

**Operation**

Shows or hides tables in the current module, if `expression` evaluates to `true` or `false`, respectively.

deleteCell, deleteColumn, deleteRow, deleteTable

**Declaration**

```dxl
string deleteCell(Object tableCell)
string deleteColumn(Object tableCell)
string deleteRow(Object tableCell)
string deleteTable(Object tableObj)
```

**Operation**

Deletes the cell, column, row, or table containing `tableCell`, which must be a table cell.

If successful, returns a null string. Otherwise, returns an error message. If the object is not a table cell, the call fails but no error is reported.

undeleteCell, undeleteColumn, undeleteRow, undeleteTable

**Declaration**

```dxl
string undeleteCell(Object tableCell)
string undeleteColumn(Object tableCell)
string undeleteRow(Object tableCell)
string undeleteTable(Object tableObj)
```

**Operation**

Undeletes the cell, column, row, or table containing `tableCell`, which must be a table cell.

If successful, returns a null string. Otherwise, returns an error message. If the object is not a table cell, the call fails but no error is reported.
for row in table

Syntax

for ro in table(Object o) do {
    
}

where:

ro is a row variable of type Object
o is an object of type Object

Operation

Assigns the cell variable ro to be each successive table row, returning row objects, which can be passed to the for cell in row loop.

for cell in row

Syntax

for co in row(Object o) do {
    
}

where:

co Is a cell variable of type Object
o Is an object of type Object

Operation

Assigns the cell variable co to be each successive row cell.

This loop returns all cells in a row regardless of whether they are displayed (filtered or deleted). To only return cells in the current display set, test each cell using isVisible(Object o).

Example 1

This outputs the identifiers of the table cells in the current table.

Object rowHead
for rowHead in table current Object do {
    Object cell
for cell in row rowHead do {
    print identifier cell "\n"
}

Example 2
This outputs the identifiers of the table cells in the current display set.

Object rowHead
for rowHead in table current Object do {
    Object cell
    for cell in row rowHead do {
        if (isVisible cell)
            print identifier cell "\n"
    }
}

Table manipulation

This section defines functions for editing and manipulating tables.

appendCell

Declaration
Object appendCell(Object tableCell)

Operation
Appends a table cell after the given object, which must be a table cell.
If the user does not have permission to create cells, or the specified object is not a table cell, a run-time error occurs.

appendColumn(table)

Declaration
Object appendColumn(Object tableCell)

Operation
Appends a table column after the given object, which must be a table cell.
If the user does not have permission to create columns, or the specified object is not a table cell, a run-time error occurs.
appendRow

Declaration

Object appendRow(Object tableCell)

Operation

Appends a table row after the given object, which must be a table cell.
If the user does not have permission to create rows, or the specified object is not a table cell, a run-time error occurs.

insertCell

Declaration

Object insertCell(Object tableCell)

Operation

Inserts a table cell before the given object, which must be a table cell.

Example

Object o = current Object
if (cell o) {
    Object newCell = insertCell o
    newCell."Object Text" = "New cell"
} else {
    ack "current object is not a cell"
}

insertColumn(table)

Declaration

Object insertColumn(Object tableCell)

Operation

Inserts a table column before the given object, which must be a table cell.

Example

Object o = current Object
if (cell o) {
    Object newColumn = insertColumn o
    newColumn."Object Text" = "New column"
} else {
    ack "current object is not a column"
}

insertRow

**Declaration**

Object insertRow(Object tableCell)

**Operation**

Inserts a table row above the given object, which must be a table cell.

**Example**

Object o = current Object
if (cell o) {
    Object newRow = insertRow o
    newRow."Object Text" = "New row"
} else {
    ack "current object is not a row"
}

getTable

**Declaration**

Object getTable(Object tableCell)

**Operation**

Returns the header object of the table containing `tableCell`. This object is not visible. It is used in calls to functions that set all the cells in a table.

ggetRow

**Declaration**

Object getRow(Object tableCell)

**Operation**

Returns the header object of the row containing `tableCell`. This object is not visible. It is used when you want to do something to all the objects in a row.
### Example

```dxl
Object tableCell
Object rowObject = getRow(aCellInTheRow)
for tableCell in rowObject do{
    // do something to the cell
}
```

### `getCellAlignment`

<table>
<thead>
<tr>
<th>Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Justification</strong></td>
</tr>
<tr>
<td><code>getCellAlignment(Object tableObject)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
</tr>
</thead>
</table>
| Returns the alignment of cells in `tableObject`.

### `getCellWidth`

<table>
<thead>
<tr>
<th>Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int getCellWidth(Object tableCell)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
</tr>
</thead>
</table>
| Returns the width in pixels of `tableCell`.

### `getCellShowChangeBars`

<table>
<thead>
<tr>
<th>Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool getCellShowChangeBars(Object tableCell)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
</tr>
</thead>
</table>
| If `tableCell` is set to show change bars, returns `true`; otherwise, returns `false`.

### `getCellShowLinkArrows`

<table>
<thead>
<tr>
<th>Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bool getCellShowLinkArrows(Object tableCell)</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>If <code>tableCell</code> is set to show link arrows, returns <code>true</code>; otherwise, returns <code>false</code>.</td>
</tr>
</tbody>
</table>
getShowTableAcrossModule

Declaration
bool getShowTableAcrossModule(Object tableCell)

Operation
If `tableCell` is set to show the table across the module, instead of just in the main column, returns `true`; otherwise, returns `false`.

setAllCellsAlignment

Declaration
void setAllCellsAlignment(Object `tableObject`, Justification `alignment`)

Operation
Sets all cells alignment within `tableObject` to have `alignment`. The `tableObject` argument must be the object returned by a call to the `getTable` function.

setAllCellsBorder

Declaration
void setAllCellsBorder(Object `tableObject`,
TableBorderPosition `edge`,
TableBorderStyle `style`)

Operation
Sets all specified border edges within `tableObject` to have the specified style.

setAllCellsShowChangeBars

Declaration
void setAllCellsShowChangeBars(Object `tableObject`,
bool `show`)

Operation
If `show` is `true`, sets the all the cells in `tableObject` to show change bars. Otherwise, sets all the cells to hide change bars. The `tableObject` argument must be the object returned by a call to the `getTable` function.
setAllCellsShowLinkArrows

**Declaration**

```c
void setAllCellsShowLinkArrows(Object tableObject,
                                bool show)
```

**Operation**

If `show` is true, sets the all the cells in `tableObject` to show link arrows. Otherwise, sets all the cells to hide link arrows. The `tableObject` argument must be the object returned by a call to the `getTable` function.

setAllCellsWidth

**Declaration**

```c
void setAllCellsWidth(Object tableObject,
                      int width)
```

**Operation**

Sets all the cells in `tableObject` to have `width` in pixels. The `tableObject` argument must be the object returned by a call to the `getTable` function.

setCellAlignment

**Declaration**

```c
void setCellAlignment(Object tableCell,
                       Justification alignment)
```

**Operation**

Sets cell alignment within `tableCell` to have `alignment`.

setCellBorder

**Declaration**

```c
void setCellBorder(Object tableCell,
                    TableBorderPosition edge,
                    TableBorderStyle style)
```

**Operation**

Sets the specified border edge to the specified style on the given cell.
setCellShowChangeBars

Declaration

void setCellShowChangeBars(Object tableCell, bool show)

Operation

If show is true, sets the cell containing tableCell to show change bars. Otherwise, sets the cell to hide change bars.

setCellShowLinkArrows

Declaration

void setCellShowLinkArrows(Object tableCell, bool show)

Operation

If show is true, sets the cell containing tableCell to show link arrows. Otherwise, sets the cell to hide link arrows.

setCellWidth

Declaration

void setCellWidth(Object tableCell, int width)

Operation

Sets the cell containing tableCell to have width in pixels.

setColumnAlignment

Declaration

void setColumnAlignment(Object tableCell, Justification alignment)

Operation

Sets the column containing tableCell to have alignment.
setColumnShowChangeBars

Declaration

void setColumnShowChangeBars(Object tableCell,
                               bool show)

Operation

If show is true, sets the column containing tableCell to show change bars. Otherwise, sets the column to hide change bars.

setColumnShowLinkArrows

Declaration

void setColumnShowLinkArrows(Object tableCell,
                              bool show)

Operation

If show is true, sets the column containing tableCell to show link arrows. Otherwise, sets the column to hide link arrows.

setColumnWidth

Declaration

void setColumnWidth(Object tableCell,
                     int width)

Operation

Sets the column containing tableCell to have width in pixels.

setRowWidth

Declaration

void setRowWidth(Object tableCell,
                  int width)

Operation

Sets the row containing tableCell to have width in pixels.
**setShowTableAcrossModule**

**Declaration**

```dxl
void setShowTableAcrossModule(Object tableCell, 
   bool showTable)
```

**Operation**

If `showTable` is true, sets the table containing `tableCell` to show the table across the module, instead of just in the main column. Otherwise, sets the table not to show across the module.

**toTable**

**Declaration**

```dxl
void toTable(Object header)
```

**Operation**

Converts a three-level object hierarchy into a table.

**Example**

This loop function detects objects that have been imported from an imaginary format called XYZ as Rational DOORS 3.0 tables, and converts them into Rational DOORS native tables.

```dxl
Object o = first current Module 
while (!null o) { 
   string importType = o."XYZ Type"
   if (!table o) { 
      if (importType == "Table") { 
         toTable o 
         o = next sibling o
      } else { 
         o = next o
      } 
   }
}
```

**Table attributes**

This section defines the functions which deal with the attributes shown in tables.

Note that the display of attributes in tables objects is controlled through the reserved “Main Column Attribute” attribute. Values of which can be assigned or obtained as per normal attributes, but with the addition of the “reserved” keyword e.g. `Object.(reserved “Main Column Attribute”) = “Object Heading”`
**useDefaultTableAttribute**

**Declaration**

```java
bool useDefaultTableAttribute(ViewDef vd)
void useDefaultTableAttribute(ViewDef vd, bool setting)
```

**Operation**

The first form returns true if the default table attribute is being used in the given view, otherwise it returns false. The second form turns the use of the default table attribute in the given view on or off.

**enableDefaultTableAttribute**

**Declaration**

```java
void enableDefaultTableAttribute(bool setting)
bool enableDefaultTableAttribute(Module)
```

**Operation**

The first form enables or disables the ability to specify a default table attribute in the current module. The second form returns true if the use of a default table attribute is enabled in the given module, otherwise it returns false.

**overrideTableAttribute**

**Declaration**

```java
void overrideTableAttribute(bool setting)
bool overrideTableAttribute(Module)
```

**Operation**

The first form sets a flag indicating that the specified default attribute for the current module should override the display attribute for all tables in the module. Setting this value will have no effect if the Default Table Attribute option is not enabled. The second form returns true if the Override Table Attribute option is enabled in the given module, otherwise it returns false.

**defaultTableAttribute**

**Declaration**

```java
void defaultTableAttribute(string AttrName)
string defaultTableAttribute(Module)
```
Operation

The first form sets the default table cell attribute on the current module. If the name provided is not a valid attribute name, then the default “Main Column” will be displayed. Setting this value will have no effect if the Default Table Attribute option is not enabled. The second form returns the name of the Default Table Attribute for the given module.

Example

//This example re-saves the current view having set the default table attribute
//to be the Object Heading, with some verification along the way.
string curViewName = currentView (current Module)
View curView = view curViewName
ViewDef vd = get curView
string MyDefTableAttr = "Object Heading"

if (!enableDefaultTableAttribute (current Module)){
    enableDefaultTableAttribute (true)
}

defaultTableAttribute (MyDefTableAttr)

if (defaultTableAttribute (current Module) != MyDefTableAttr){
    print "An error occurred setting the default table attribute.\n"
} else {
    useDefaultTableAttribute (vd, true)

    if (!useDefaultTableAttribute (vd)){
        print "An error occurred while activating the default table attribute on
the current view.";
    } else {
        save (curView, vd)
    }
}
Chapter 32
Rich text

This chapter describes the functions that allow manipulation of rich text.

- Rich text processing
- Rich text strings
- Enhanced character support
- Importing rich text
- Diagnostic perms

Rich text processing

This section gives the syntax for operators, functions and a for loop, which can be used to process rich text. These elements use internal data types, so declarations are not stated.

A rich text string contains sections of formatting, referred to as chunks. Each chunk can be processed using the core for loop that performs the decomposition. Chunks are processed as variables of type RichText, from which different properties can be extracted.

These decomposition functions are particularly valuable for implementing exporters that have to generate formatting information.

Rich text tags

The following tags can be used in DXL code to create rich text strings:

\b bold text
\i italic text
\ul underlined text
\strike struck through text
\sub subscript text
\super superscript text
\nosupersub neither subscript nor superscript

The syntax for using these tags within a string is as follows:

{tag<space>text}
or
{tag{text}}

Tags can be nested, to apply more than one type of formatting, as follows:
{tag<space>text{tag<space>text}}}

**Note:** Remember that the back-slash character (\) must be escaped with another back-slash character in a string.

---

### Rich text constructors

The dot operator (.) is used to extract information from rich text chunks.

**Syntax**

`richString.richTextProperty`

where:

- `richString` Is a chunk of rich text of type RichText
- `richTextProperty` Is one of the properties described below

**Operation**

The properties act on the chunk of rich text as follows:

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>The text of a chunk of rich text as a string without formatting</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boolean property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>bold</td>
<td>Whether the chunk of rich text has bold formatting</td>
</tr>
<tr>
<td>last</td>
<td>Whether the chunk of rich text is the last in the string</td>
</tr>
<tr>
<td>italic</td>
<td>Whether the chunk of rich text has italic formatting</td>
</tr>
<tr>
<td>newline</td>
<td>Whether the chunk of rich text is immediately followed by a newline character</td>
</tr>
<tr>
<td>strikethru</td>
<td>Whether the chunk of rich text has strike through formatting</td>
</tr>
<tr>
<td>subscript</td>
<td>Whether the chunk of rich text has subscript formatting</td>
</tr>
</tbody>
</table>
### richText(column)

**Declaration**

```dxl
def richText(Column c, Object o):
```

**Operation**

Returns the text contained in column `c` for the object `o` as rich text.

### richTextWithOle(column)

**Declaration**

```dxl
def richTextWithOle(Column c, Object o):
```

**Operation**

Returns the text contained in column `c` for the object `o` as rich text, including OLE objects.

### richTextWithOleNoCache(column)

**Declaration**

```dxl
def richTextWithOleNoCache(Column c, Object o):
```

**Operation**

Returns the text contained in column `c` for the object `o` as rich text, including OLE objects, and clears the OLE cache.

### richTextNoOle(column)

**Declaration**

```dxl
def richTextNoOle(Column c, Object o):
```

For examples, see the **for rich text in string loop**.

<table>
<thead>
<tr>
<th>Boolean property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>superscript</td>
<td>Whether the chunk of rich text that has superscript formatting</td>
</tr>
<tr>
<td>underline</td>
<td>Whether the chunk of rich text that has underline formatting</td>
</tr>
</tbody>
</table>

Boolean property Extracts

Whether the chunk of rich text that has superscript formatting

Whether the chunk of rich text that has underline formatting
Operation

Returns the text contained in column c for the object o as rich text, excluding OLE objects.

---

**removeUnlistedRichText**

**Declaration**

```dxl
string removeUnlistedRichText(string s)
```

**Operation**

Removes rich text markup that Rational DOORS does not recognize. Fonts are preserved when importing Word or RTF documents. Fonts can be specified by inserting a symbol from a specific font.

**Example**

This example prints `{\b bold text}` in the DXL Interaction window’s output pane:

```dxl
string s = "{\b \unknown bold text}"
print removeUnlistedRichText s
```

---

**for rich text in string**

**Syntax**

```dxl
for rt in string s do {
    ...
}
```

**where:**

- `rt` is a variable of type `RichText`
- `s` is a string containing valid rich text

**Operation**

Assigns the rich text variable `rt` to be each successive chunk of formatting in a rich text string, returning each as a pointer to a structure of type `RichText`. This structure can tell you whether a piece of text is bold, italic, underlined, struck through, subscript, superscript, or at the end of a line.

**Example**

```dxl
string s = "{\b Bold}{\i Italic}DXL"
RichText rt
for rt in s do {
    if (rt.italic) print rt.text " is italic\n"
    else if (rt.bold) print rt.text " is bold\n"
    else print rt.text " is neither\n"
```
This example prints:

- Bold is bold
- Italic is italic
- DXL is neither

**RichTextParagraph type properties**

Properties are defined for use with the . (dot) operator and a RichTextParagraph type handle to extract information from a RichTextParagraph type, as shown in the following syntax:

**Syntax**

```dxl
for <RichTextParagraph> in <string> do
```

**Operation**

Loops through the rich text paragraph RichTextParagraph in the string string.

The following tables list the properties and the information they extract:

### Integer property | Extracts
---|---
indentLevel | The indent level of the paragraph. The units are twips (= 1/20 point or 1/1440 inch). Currently the base unit of indentation in Rational DOORS is 360 twips, so values of indentLevel will be multiples of 360.
bulletStyle | The bullet style, as an integer. Currently the only values are 0 (no bullets) and 1 (bullets).

### Boolean property | Extracts
---|---
isBullet | Whether the paragraph has a bullet point.

### String property | Extracts
---|---
text | The plain text of the paragraph.

**Example**

```dxl
void dumpParagraphs(string s)
{
    RichTextParagraph rp

    for rp in s do {
```

DXL Reference Manual
RichText type properties

Properties are defined for use with the . (dot) operator and a RichText type handle to extract information from a RichText type, as shown in the following syntax:

**Syntax**

```dxl
for <RichText> in <RichTextParagraph> do
```

**Operation**


The following tables list the properties and the information they extract:

<table>
<thead>
<tr>
<th>Integer property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>indentLevel</td>
<td>The indent level of the rich text chunk. The units are twips (= 1/20 point or 1/1440 inch). Currently the base unit of indentation in Rational DOORS is 360 twips, so values of indentLevel will be multiples of 360. The value will remain the same for all chunks in a line.</td>
</tr>
<tr>
<td>bulletStyle</td>
<td>The bullet style, as an integer. Currently the only values are 0 (no bullets) and 1 (bullets). The value will remain the same for all chunks in a line.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boolean property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>isBullet</td>
<td>Whether the paragraph has a bullet point. The value will remain the same for all chunks in a line.</td>
</tr>
<tr>
<td>isUrl</td>
<td>Whether the rich text chunk is a URL.</td>
</tr>
<tr>
<td>isOle</td>
<td>Whether the rich text chunk represents an OLE object.</td>
</tr>
</tbody>
</table>
```
Example

void dumpAllInfo(RichText rt)
{
    print "***********New chunk:\n"
    print "text:" rt.text "::  "
    print "bold:" rt.bold "::  "
    print "italic:" rt.italic "::  "
    print "underline:" rt.underline "::\n"
    print "strikethru:" rt.strikethru "::  "
    print "superscript:" rt.superscript "::  "
    print "subscript:" rt.subscript "::\n"
    print "charset:" rt.charset "::\n"
    print "newline:" rt.newline "::  "
    print "last:" rt.last "::\n"
    // new in 6.0
    print "isOle:" rt.isOle "::  "
    print "indent:" rt.indentLevel "::  "
    print "bullet:" rt.isBullet "::  "
    print "bulletStyle:" rt.bulletStyle "::  "
    print "isUrl:" rt.isUrl "::\n"
}

void dumpAllParagraphs(string s)
{
    RichTextParagraph rp
    RichText rt

    for rp in s do {
        print "****New paragraph\n"
        print "text:" rp.text "::\n"
        print "indent:" rp.indentLevel "::  "
        print "bullet:" rp.isBullet "::\n"
        print "bulletStyle:" rp.bulletStyle "::\n"
        for rt in rp do
            dumpAllInfo rt
    }
}
```
Object o = current
string s = richTextWithOle o."Object text"

dumpAllParagraphs s

---

Rich text strings

This section defines an operator and functions for strings containing rich text.

Assignment (rich text)

The equals operator (=) is used to assign rich text format to attributes, as follows:

Syntax

\[ \text{attrRef} = \text{richText(string s)} \]

where \( \text{attrRef} \) can be one of:

(\text{Object o}).(\text{string attrName})

(Module m).(\text{string attrName})

(Link l).(\text{string attrName})

where:

\( o \) is an object of type Object

\( m \) is a module of type Module

\( l \) is a link of type Link

\( \text{attrName} \) is a string identifying the attribute

Operation

Sets the attribute called \( \text{attrName} \) to the rich text string contained in \( s \).

Example

Object o = current

\( o."Object Text" = \text{richText } \"\{\backslash b \text{ BOLD}\}\" \)

\( o."Object Heading" = \"\{\backslash b \text{ BOLD}\}\" \)

This sets:

- The current object’s text to BOLD
- The current object’s heading to \( \{\backslash b \text{ BOLD}\} \) which is displayed as \{\textbf{bold}\}
This demonstrates the importance of using the richText function in both getting and setting attribute values if you wish to maintain the rich text content. If you do not process the string value with richText, all the markup is escaped with backslashes and becomes apparent to the user.

**cutRichText**

**Declaration**

```cpp
string cutRichText(string s,
                   int start,
                   int end,
)
```

**Operation**

Returns the string s with the displayed characters from start to end removed. For the purposes of counting characters, rich text markup is ignored, and markup is preserved.

**Example**

```cpp
cutRichText("{\b 0123456}", 1, 3)
```

This example returns: `{\b 0456`

**findRichText**

**Declaration**

```cpp
bool findRichText(string s,
                   string sub,
                   int& offset,
                   int& length,
                   bool matchCase)
```

**Operation**

Returns true if string s contains the substring sub.

If matchCase is true, string s must contain string sub exactly with matching case; otherwise, any case matches.

The function returns additional information in offset and length. The value of offset is the number of characters in s to the start of the first match with string sub. The value of length contains the number of characters in the matching string. The function replaceRichText uses offset and length to replace the matched string with another string.

**Example**

```cpp
string s = "{\b This is Bo{\i ld and italic}}" 
string sub = "bold"
int    offset 
int    len 
```
if (findRichText(s, sub, offset, len, false)) {
    print "Offset = " offset "\n"
    print "Length = " len "\n"
} else {
    print "Failed to match"
}

This example prints:
Offset = 12
Length = 8
because the braces are delimiters, not characters in the string.

---

### isRichText

**Declaration**

```cpp
bool isRichText(string s)
```

**Operation**

Returns **true** if string `s` is in the Rational DOORS rich text format; otherwise, returns **false**.

If **false** is returned, `s` cannot be used to set any object attribute value.

**Example**

This example prints **true** in the DXL Interaction window’s output pane:

```dxl
print isRichText "\{\i correct balance\}"
```

This example prints **false** in the DXL Interaction window’s output pane:

```dxl
isRichText "\{\b missing bracket"
```

---

### replaceRichText

**Declaration**

```cpp
string replaceRichText(string s,
                        int offset,
                        int length,
                        string r)
```

**Operation**

Returns a string, which is equivalent to `s` but with the characters between `offset` and `offset+length` replaced with `r`, whilst retaining formatting tags.

**Example**

```dxl
RichText rt
string s = "\{\b This is Bo\{\i ld and italic\}\}"
```
string r = "bOLD"
string result = replaceRichText(s, 12, 8, r)
print result "\n"
    Prints:
   \{b This is bO{i LD and italic}\}

richtext_identifier(Object)

Declaration

string richtext_identifier(Object o)

Operation

Returns the object identifier (which is a combination of module prefix and object absno) as an RTF string.

Example

Object o = current
print richtext_identifier(o)

pasteToEditbox

Declaration

bool pasteToEditbox()

Operation

Pastes the contents of the clipboard into a module object that is ready for in-place editing. If the paste fails, the function returns false.

Example

This example pastes bold text to an open module:
setRichClip richText "\{\b bold text\}"
pasteToEditbox

richClip

This function has the following syntax:
richClip()

Gets the rich text contents of the system clipboard as a rich text string.

Example

o."Object Text" = richClip
setRichClip

**Declaration**

```c
void setRichClip(richText(string s)
                [, string styleName])
```

**Operation**

Sets the system clipboard to contain the rich text string `s`. Optionally, you can include a minimal RTF style sheet that contains a supplied style name, which sets the string style.

**Example**

```c
setRichClip richText o."Object Text"
// with style sheet
setRichClip(richText o."Object Heading,
           "Heading 1")
```

setRichClip(Buffer/RTF_string__)

**Declaration**

```c
void setRichClip(RTF_string__ s, string styleName, string fontTable)
void setRichClip(Buffer buff, string styleName, string fontTable)
void setRichClip(RTF_string__ s, string styleName, string fontTable, bool keepBullets, bool keepIndents)
void setRichClip(Buffer buff, string styleName, string fontTable, bool keepBullets, bool keepIndents)
```

**Operation**

First form sets the system clipboard with the rich text obtained by applying the style `styleName` to the string `s`, using the font table `fontTable` supplied, which should include a default font. Font numbers in the string `s` will be translated to the supplied font table `fontTable`.

Second form is same as the first but the source is a buffer `buff` rather than an `RTF_string__`.

Third form sets the system clipboard with the rich text obtained by applying the style `styleName` to the string `s`, using the font table `fontTable` supplied. If `keepBullets` is `false`, any bullet characters are removed from string `s`. If `keepIndents` is `false`, any indentation is removed from string `s`. If `keepBullets` and `keepIndents` are both `true`, the behavior is exactly the same as the first form.

Fourth form is same as the third but the source is a buffer `buff` other than an `RTF_string__`.

**Example 1**

The following code:

```c
string s = "hello"
```
string fontTable = "\\deff0{\\fonttbl {\\f1 Times New Roman;}}"
setRichClip(richText s, "Normal", fontTable)
puts the following rich text string onto the system clipboard:

{\rtf1 \deff0{\fonttbl {\f1 Times New Roman;}}{\stylesheet {\s1 Normal;}}{\s1 hello\par}}

Example 2

string bulletedString =
"{\\rtf1\ansi\ansicpg1252\deff0\deflang1033{\\fonttbl{\\f0\\fswiss\\fcharset0 Arial;}{\\f1\\fnil\\fcharset2 Symbol;}}
\\viewkind4\\uc1\\pard\\f0\\fs20 Some text with\par\\pard\(\pntext\\f1\'B7\tab\{\*\pn\pn1vlblt\pnf1\pindent0\{\\pntxtb\'B7 \}}\\fi-720\\li720 bullet 1\par
\\pntext\\f1\'B7\tab\{bullet 2\}\\par
\\pard bullet points in it.\\par
\\par
}"

string fontTable = "\\deff0{\\fonttbl{\\f0\\fswiss\\fcharset0 Arial;}{\\f1\\fnil\\fcharset2 Symbol;}}"
setRichClip(richText bulletedString, "Normal", fontTable)

// the previous call puts
// "{\\rtf1 \deff0{\\fonttbl{\\f0\\fswiss\\fcharset0 Arial;}{\\f1\\fnil\\fcharset2 Symbol;}}{\\stylesheet {\\s1 Normal;}}{\\s1 Some text with\par {\\f1\'b7\tab\{bullet 1\par {\\f1\'b7\tab\{bullet 2\par bullet points in it.\par \par}}
\\par
}"

setRichClip(richText bulletedString, "Normal", fontTable, false, false)

// the previous call puts
// "{\\rtf1 \deff0{\\fonttbl{\\f0\\fswiss\\fcharset0 Arial;}{\\f1\\fnil\\fcharset2 Symbol;}}{\\stylesheet {\\s1 Normal;}}{\\s1 Some text with\par bullet 1\par bullet 2\par bullet points in it.\par \par}}"

// on the clipboard

setRichClip(richText bulletedString, "Normal", fontTable, false, false)

// the previous call puts
// "{\\rtf1 \deff0{\\fonttbl{\\f0\\fswiss\\fcharset0 Arial;}{\\f1\\fnil\\fcharset2 Symbol;}}{\\stylesheet {\\s1 Normal;}}{\\s1 Some text with\par bullet 1\par bullet 2\par bullet points in it.\par \par}}"

// on the clipboard -- note no bullet symbols (\'b7) in the markup
**rtfSubString**

**Declaration**

Buffer rtfSubString(Buffer input, Buffer output, int start, int end)

**Operation**

This function takes an RTF buffer and returns the RTF sub-string between the defined start and end points.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>The complete RTF text. This can be full RTF or an RTF fragment, but must be valid RTF and not plain text.</td>
</tr>
<tr>
<td>output</td>
<td>The buffer in which the sub-string will be returned. This buffer must be created before calling rtfSubString. This return value will always be full RTF. A reference to this buffer is the return value of the function.</td>
</tr>
<tr>
<td>start</td>
<td>The zero-based start point of the sub-string.</td>
</tr>
<tr>
<td>end</td>
<td>The end point of the sub-string.</td>
</tr>
</tbody>
</table>

**Example**

Object o = current
Buffer input = create
Buffer output = create
input = o."Object Text"
rtfSubString(input, output, 4, 8)
print stringOf(output)

**richText(of attribute)**

**Declaration**

Buffer richText(attrRef

   [,bool includeFontTable])

where attrRef can be one of:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)
where:

\[ o \quad \text{is an object of type Object} \]
\[ m \quad \text{is a module of type Module} \]
\[ l \quad \text{is a link of type Link} \]
\[ attrName \quad \text{is a string identifying the attribute} \]

**Operation**

Returns the rich text version of an attribute called \( attrName \), if \( \text{includeFontTable} \) is \( \text{false} \), or not present.

If the boolean argument is \( \text{true} \), it returns the rich text version of an attribute value appended to the font table for that module. The Boolean argument is only applicable to the string version.

This preserves the meaning of font markup when moving rich text attribute values between modules.

**Example**

```dxl
print richText (current Object) . "Object Text"
```

If the Object text attribute of the current object is \textbf{Engine}:

```dxl
{\b Engine }
```

Module oldm, newm
Object oldo, newo
oldo = first oldm
newo = create newm
newo."Object Text" = richText(oldo."Object Text", true)

**Example 2**

Object o  = current Object
Buffer b = create
b = richText(o."Object Text")
print stringOf b
delete b
```

**richText(of string)**

**Declaration**

```dxl
string richText(string s)
```

**Operation**

Returns a string, which is the correct rich text version of string \( s \). It inserts a backslash escape character before unescaped braces and unescaped backslashes. This makes it suitable for assignment to attribute values.
Example

print richText "\{ \hello \}"
Prints:
"\{ \hello \}"

string exportAttributeToFile

Declaration

string exportAttributeToFile(attrRef, string fileNameWithCompletePath)
where attrRef can be one of:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation

Writes the rich text of attribute attr, including the OLE objects to file fileNameWithCompletePath. If fileNameWithCompletePath does not already exist, it is created. If it already exists, it is overwritten.

Returns null on success, or an error message on failure.

stringOf(rich text)

Declaration

string stringOf(richText(string s))

Operation

This enables access to rich text as a string.

richTextWithOle

Declaration

string richTextWithOle(attrRef attr)
where attrRef is in one of the following formats:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)
Operation
Returns the rich text of attribute attr, including the OLE objects. The use of this perm should be confined to copying rich text values from one attribute to another.

richTextWithOleNoCache

Declaration
string richTextWithOleNoCache(attrRef attr)
where attrRef is in one of the following formats:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation
Returns the rich text of attribute attr, including the OLE objects, and clears the OLE cache. The use of this perm should be confined to copying rich text values from one attribute to another.

richTextNoOle

Declaration
string richTextNoOle(attrRef attr)
where attrRef is in one of the following formats:
(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation
Returns the rich text of attribute attr, excluding the OLE objects. The use of this perm should be confined to copying rich text values from one attribute to another.

applyTextFormattingToParagraph

Declaration
string applyTextFormattingToParagraph(string s, bool addBullets, int indentLevel, int paraNumber, [int firstIndent])

Operation
Applies bullet and/or indent style to the given text, overwriting any existing bulleted/indenting.
• If addBullets is true, adds bullet style.
• If `indentLevel` is nonzero, adds indenting to the value of `indentLevel`. The units for `indentLevel` are twips = twentieths of a point.

• If `paraNumber` is zero, the formatting is applied to all the text. Otherwise it is only applied to the specified paragraph number.

• If the optional parameter `firstIndent` is specified, then this sets the first line indent. If the value is negative then this sets a hanging indent. The units are in points.

The input string `s` must be rich text. For example, from `string s = richText o."Object Text"`. Returns a rich text string which describes the text with the formatting applied.

**Example**

Object o = current
string s = o."Object text"
o."Object text" = richText (applyTextFormattingToParagraph(richText s, true, 0, 0))

Adds bullet style to all of the current object’s text.

---

**exportRTFString**

**Declaration**

string exportRTFString(string text)

**Operation**

Translates a Rational DOORS rich text string to the RTF standard. Newlines are converted into `\par` tags, not `\newline`.

For use with legacy RTF only; any new code should use `removeUnlistedRichText`.

**Example**

Object o = current
string str = o."Object Text"
string rtf_string = exportRTFString(str)

---

**richTextFragment**

**Declaration**

string richTextFragment(string richString)

string richTextFragment(string richString, string fontTable)

string richTextFragment(string richText [, string fontTable [, bool inTable]])
Operation

The first form takes an argument \texttt{richFrag} which should be a rich text string. Returns an equivalent representation of the rich text with RTF header information removed. Useful for building up a real RTF string without having to cope with the header information every time. Font markup will be mapped to the Rational DOORS default font table.

The second form is the same as the first except for the second argument, \texttt{fontTable}, which is a font table string. Any font markup in the string is mapped to the first font in the font table passed in which has the same character set as the original markup.

The third form has an optional boolean argument which, if provided and set to \texttt{true}, ensures that the returned rich text string is valid as contents of a rich text table. Use this setting if multiple paragraphs are exported to a single table cell and the resulting rich text output is to be opened by MS-Word 2000.

Example 1

```dxl
string richString = 
"{\rtf1\ansi\ansicpg1252\deff0\deflang1033
{\fonttbl{\f0\fnil\fprq1\fcharset0 Times New Roman;}
{\f1\froman\fprq2\fcharset0 Symbol;}}
{\colortbl ;\red0\green0\blue0;}
\viewkind4\uc1\pard\cf1\f0\fs20
Some plain text.
\b Some bold text.\b0
\b\i Some bold and italic text. \b0\i0
A symbol \f1 a\f0 (alpha).\b\i\par }"
```

print richTextFragment richString

// returns

// Some plain text. \b Some bold text.\b0 \b \i Some bold and italic text. \b0 \i0
\b0 \i0 A symbol \f1001 a\f1007 (alpha).\}

Example 2

```dxl
string fontTable = 
"\\deff1\\fonttbl
{\\f0\\fswiss\\fcharset0 Arial;}
{\\f1\\froman\\fprq2\\fcharset0 Times New Roman;}
{\\f2\\froman\\fprq2\\fcharset2 Symbol;}"
```

print richTextFragment( richString, fontTable)

// returns

// Some plain text. \b Some bold text.\b0   \b \i Some bold and italic text. \b0 \i0  A symbol \{\f2 a}\{\f0 (alpha).\}

Enhanced character support

This section lists constants and defines functions for the display and printing of characters outside the ANSI character set.

Character set constants

For the display and printing of characters outside the ANSI character set, you can specify another character set. For the results to be correct, you must have the appropriate fonts installed. The following integer character set constants are declared:

charsetAnsi
charsetSymbol
charsetGreek
charsetRussian
charsetEastEurope
charsetTurkish
charsetHebrew

Certain UNIX machines may not display some characters correctly.

For an example of the use of character set constants, see "Character set identification" below.

Character set identification

The dot operator (.) is used to identify the character set of rich text, as follows:

Syntax

richString.charset()

where:

richString is a chunk of rich text of type RichText

Operation

Returns the character set of a chunk of rich text.
Example

for rt in s do {
    if (rt.charset == charsetAnsi) {
        print rt.text " is in the ANSI character set\n"
    } else if (rt.charset == charsetSymbol) {
        print rt.text " is in the Symbol character set\n"
    } else {
        print rt.text " is in character set number " rt.charset "\n"
    }
}

charsetDefault

Declaration

int charsetDefault()

Operation

Returns the system default character set. On UNIX platforms, this is always charsetAnsi. On Windows systems, this is the user's local setting.

Example

string s
RichText rt
for rt in s do {
    if (rt.charset == charsetAnsi) {
        print rt.text " is in the ANSI character set\n"
    } else if (rt.charset == charsetSymbol) {
        print rt.text " is in the Symbol character set\n"
    } else {
        print rt.text " is in the character set number " rt.charset "\n"
    }
    if (rt.charset == charsetDefault) {
        print rt.text " is in your system default character set\n"
    }
}
characterSet

Declaration

void characterSet(DBE canvas,
    int level,
    int mode,
    int characterSet)

Operation

Sets the level, mode and character set for drawing strings on the canvas. Through the font tables, this sets the font.

Example

DB symbolBox = create "Symbols"
void repaint(DBE symbol) {
    int fsize = 1 // level 1 size
    int mode = 0 // body text style
    background(symbol,
        logicalPageBackgroundColor)
    color(symbol, logicalDataTextColor)
    characterSet(symbol, fsize, mode,
        charsetAnsi)
    draw(symbol, 10, 20, "abc")
    // appears as abc
    characterSet(symbol, fsize, mode,
        charsetSymbol)
    draw(symbol, 40, 20, "abc")
    // appears as alpha beta chi
}
DBE symbol = canvas(symbolBox, 100, 50, repaint)
show symbolBox

fontTable

Declaration

string fontTable(Module m)

Operation

Returns the module’s font table, which is used for mapping rich text font markup to character set information.
Example

print fontTable current Module
In a newly created module, the Rational DOORS default font table is:

{\f1016\fswiss\fcharset134 Tahoma;}
{\f1015\fswiss\fcharset136 Tahoma;}
{\f1014\fswiss\fcharset129 Tahoma;}
{\f1013\fswiss\fcharset128 Tahoma;}
{\f1012\fswiss\fcharset177 Arial;}
{\f1011\fswiss\fcharset162 Arial;}
{\f1010\fswiss\fcharset238 Arial;}
{\f1009\fswiss\fcharset204 Arial;}
{\f1008\fswiss\fcharset161 Arial;}
{\f1007\fswiss\fcharset0 Arial;}
{\f1006\froman\fcharset177 Times New Roman;}
{\f1005\froman\fcharset162 Times New Roman;}
{\f1004\froman\fcharset238 Times New Roman;}
{\f1003\froman\fcharset204 Times New Roman;}
{\f1002\froman\fcharset161 Times New Roman;}
{\f1001\ftech\fcharset2 Symbol;}
{\f1000\froman\fcharset0 Times New Roman;}

---

Importing rich text

This section defines a function for importing rich text.

importRTF

Declaration

int importRTF(string file,
               Module m,
               bool mapStyles,
               bool dynamicUpdate)
Operation

Imports the rich text format file file, into a new sibling at the same level as the current object of module m. If mapStyles is set, you are prompted to match styles if non-standard styles are used. If dynamicUpdate is set, the displayed module is refreshed.

Returns

#define ecOK 0       /* Everything's fine */
#define ecStackUnderflow 1   /* Unmatched '}' */
#define ecStackOverflow 2      /* Too many '{' -- memory exhausted */
#define ecUnmatchedBrace 3     /* RTF ended during an open group */
#define ecInvalidHex 4         /* invalid hex character found in data */
#define ecBadTable 5           /* RTF table (sym or prop) invalid */
#define ecAssertion 6          /* Assertion failure */
#define ecEndOfFile 7          /* end of file reached */
#define ecFileNotFound 8       /* The file could not be found (or opened) */

Example

Int i = importRTF("c:\doors\examples\parse.rtf", current Module, false, false)
if (i == 0)
{
    print "Successful\n"
} else {
    print "Failed - return code " i " \n"
}

Diagnostic perms

These perms are for run-time richText/OLE DXL diagnostics. DXL scripts written for pre-V6 Rational DOORS do not specify whether OLE objects should be included in richText extracted from Object Text attributes or the main column in a view. If diagnostics are enabled, the user can be given warnings when this occurs, enabling the user to replace the richText() call with richTextWithOle() or richTextNoOle(). The user can also be warned when a new value is assigned to an Object Text attribute, as this will now replace any OLE objects in the Object Text.

enableObjectTextAssignmentWarnings

Declaration

enableObjectTextAssignmentWarnings(string logFile)
Operation

Enables warnings whenever a new value is assigned to an Object Text attribute. Warnings are disabled by default. This perm returns no value. The logFile argument enables the user to specify a file where filenames and line numbers will be logged, where warnings are issued. If this argument is null, no logging is done. If the file cannot be opened, a warning message is displayed. If a log file has already been opened, this argument has no effect.

disableObjectTextAssignmentWarnings

Declaration
disableObjectTextAssignmentWarnings()

Operation

Disables warnings whenever a new value is assigned to an Object Text attribute. Warnings are disabled by default. This perm returns no value.

enableObjectTextRichTextWarnings

Declaration

enableObjectTextRichTextWarnings(string logFile)

Operation

Enables warnings whenever the richText (Attribute) perm is applied to an Object Text attribute, or the richText (Column, Object) perm is applied to the Main column. The logFile argument is treated the same way as that for enableObjectTextAssignmentWarnings(). If a logFile has already been opened, this argument has no effect.

This perm returns no value.

disableObjectTextRichTextWarnings

Declaration

disableObjectTextRichTextWarnings()

Operation

Disables warnings when the richText (Attribute) perm is applied to an Object Text attribute, or the richText (Column, Object) perm is applied to the Main column. If enableGeneralRichTextWarnings() has been called, warnings will still be issued for all richText() perms, until disableGeneralRichTextWarnings() is called.

This perm returns no value.
enableGeneralRichTextWarnings

Declaration

enableGeneralRichTextWarnings(string logFile)

Operation

Enables warnings whenever the richText (Attribute) or richText (Column, Object) perm is called. The logFile argument is treated the same way as for the other enable perms above. This perm returns no value.

disableGeneralRichTextWarnings

Declaration

disableGeneralRichTextWarnings()

Operation

Normally disables warnings whenever the richText (Attribute) or richText (Column, Object) perm is called. The exception to this is if enableObjectTextRichTextWarnings() has been called, warnings will still be issued when these richText perms are applied to Object Text or the Main column. This perm returns no value.

enableRepeatWarnings

Declaration

enableRepeatWarnings()

Operation

Enables multiple repeated warnings to be issued for the same DXL script file/line-number combination, whenever that code is executed by the interpreter. By default, only one warning is issued for any file/line in any one Rational DOORS client session. This perm returns no value.

disableRepeatWarnings

Declaration

disableRepeatWarnings()

Operation

This perm negates the effect of enableRepeatWarnings(). It returns no value. Note that the repeat prevention does not apply to DXL scripts run from the DXL Interaction window.
**disableDisplayWarnings**

**Declaration**

disableDisplayWarnings()

**Operation**

Disables the pop-up warning dialogs. If enabled, warnings are still logged in the specified logFile. This perm returns no value.

**enableDisplayWarnings**

**Declaration**

enableDisplayWarnings()

**Operation**

Enables pop-up warning dialogs. It returns no value.

**dxlWarningFilename**

**Declaration**

string dxlWarningFilename()

**Operation**

Returns the filename quoted in the last pop-up warning dialog.

**dxlWarningLineNumber**

**Declaration**

int dxlWarningLineNumber()

**Operation**

Returns the line number quoted in the last pop-up warning dialog.
Chapter 33
Spelling Checker

This chapter describes the following features of the spelling checker:

- Constants and general functions
- Language and Grammar
- Spelling Dictionary
- Miscellaneous Spelling
- Spelling\Dictionary Examples

Constants and general functions

Language Constants

**Operation**
The following are used to specify one of the standard supported languages:

- USEnglish
- UKEnglish
- French
- German
- GermanReform

**Example**

```plaintext
SpellingOptions options
getOptions(options, userSpellingOptions)
setLanguage(options, German)
saveOptions(options)
```

Options Constants

**Operation**
The following are used by the `getOptions` function to specify which set of spelling options are to be opened:

databaseSpellingOptions
userSpellingOptions

Example
SpellingOptions options
g_OPTS(options, databaseSpellingOptions)

Dictionary Constants

Operation
The following are used by the open function to indicate which type of dictionary is to be opened:
databaseDictionary
clientDictionary

Example
Dictionary d
do(d, clientDictionary)
insert(d, "IBM")

Grammar Constants

Operation
The following are used to define the formality of grammar checking:
informalGrammar
standardGrammar
formalGrammar

Example
SpellingOptions options
g_OPTS(options, userSpellingOptions)
setGrammarLevel(options, informalGrammar)
s_Options(options)

Spell Check Mode Constants

Operation
The following are used to define the level of spell checking to be carried out:
speelingOnly
quickProof
fullProof

Example
SpellingOptions options
getOptions(options, userSpellingOptions)
setCheckMode(options, quickProof)

spell

Declaration
string spell(string word)
string spell(Object o,
            string attrName,
            int &start,
            int &end)

Operation
The first form checks the word for spelling, and returns a null string if it is correct or if word is a null string. If the spelling is not correct, returns an error message.

The second form checks the attribute name for spelling, and returns a null string if it is correct, if attrName is a null string, or if the specified attribute is not contained in the specified object. If the spelling is not correct, returns an error message. It only works with string or text attributes.

The start and finish arguments must be initialized to zero before the function is called. If the contents of attrName are misspelled, the function sets the values of start and finish to identify the first and last characters of the incorrectly spelled substring.

Example
Object o = current
// check status
if (o != null)
{
    int iStart = 0, iFinish = 0
    // get attribute info
    string sObjectHeading = probeRichAttr_(o,
                                      "Object Heading")
    int iLength = length(sObjectHeading)
    // process attribute
    while (iStart < iLength)
    {
        // check attribute
if (spell(o, "Object Heading", iStart, iFinish) != null)
{
    // warn user
    print "Spelling mistake located [" 
    print iStart ":" iFinish "] - '
    print sObjectHeading[iStart:iFinish] "\n"
    // adjust accordingly
    iStart = iFinish
}
}

spellFix

**Declaration**

string spellFix(Object o,
    string attrName,
    int &start,
    int &end,
    string newString)

**Operation**

Replaces a misspelled string within the specified attribute, which must be a string or text attribute. The string is identified using `start` and `finish`, provided the `spell` function has previously been called on the object and attribute.

In cases where the new string is a different length from the misspelled substring, the function resets the values of `start` and `finish`.

Returns a null string if it the substring is replaced successfully or if the specified attribute is not contained in the specified object. Otherwise, returns an error message.

alternative

**Declaration**

string alternative(int n)

**Operation**

Returns the `n`th spelling for the word last passed to `spell`.

alternatives

**Declaration**

int alternatives()
Operation

Returns the number of options found for the last call to spell.

for all spellings

Syntax

for s in alternatives do {
  ...
}

where:

s is a string variable

Operation

Assigns string s to be each successive value found for the last spelling check.

Example

string mess = spell("whta")

if (null mess) {
  print "You are a spelling bee\n"
} else {
  int n = alternatives
  print "There are " n " other spellings:\n"
  string altSpelling
  for altSpelling in alternatives do
    print altSpelling "\n"
}

spell

Declaration

Buffer spell(Buffer returnBuffer, Buffer word)

Buffer spell(Buffer returnBuffer,
  Object o,
  string attribute,
  int &wordStart,
  int &wordEnd,
  int &sentenceStart,
  int &sentenceEnd,
  int &ruleType,
  bool spellingErrorsFirst,
  bool &deletionError)
Operation

First form of the spell perm that returns a buffer \textit{returnBuffer}, to reduce memory usage caused by using strings. The perm returns an empty Buffer if the \textit{word} is correct, or an error message otherwise.

- \textit{returnBuffer}
  
  Buffer used to create the return value - must be created before calling.

- \textit{word}
  
  The word to be checked.

The second form of this perm checks spelling and grammar in the named attribute \textit{attribute} of the specified object \textit{o}. If an error is found, the error details returned in the parameters relate to the first error. Call \texttt{getNextError()} to view subsequent error details.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{returnBuffer}</td>
<td>Buffer used to create the return value - must be created before calling.</td>
</tr>
<tr>
<td>\textit{o}</td>
<td>The object to be checked.</td>
</tr>
<tr>
<td>\textit{attribute}</td>
<td>Name of the specific attribute to be checked.</td>
</tr>
<tr>
<td>\textit{wordStart}</td>
<td>If an error is found, returns the start position of the incorrect word.</td>
</tr>
<tr>
<td>\textit{wordEnd}</td>
<td>If an error is found, returns the end position of the incorrect word.</td>
</tr>
<tr>
<td>\textit{sentenceStart}</td>
<td>If an error is found, returns the start position of the sentence containing the error.</td>
</tr>
<tr>
<td>\textit{sentenceEnd}</td>
<td>If an error is found, returns the end position of the sentence containing the error.</td>
</tr>
<tr>
<td>\textit{ruleType}</td>
<td>If an error is found, returns the code of the rule that triggered the error.</td>
</tr>
<tr>
<td>\textit{spellingErrorsFirst}</td>
<td>Specifies whether spelling errors should be reported before grammar errors - note that this operates at a sentence level.</td>
</tr>
<tr>
<td>\textit{deletionError}</td>
<td>If an error is found, this flag indicates that the error type recommends that text is deleted (e.g. this will occur when a word is duplicated, such as &quot;This is is an error.&quot;)</td>
</tr>
</tbody>
</table>

\texttt{getNextError}

### Declaration

```dxl
void getNextError(Buffer errorString,
                  int &wordStart,
                  int &wordEnd,
                  int &sentenceStart,
                  int &sentenceEnd,
```

DXL Reference Manual
bool &correctionComplete,
bool skipSentence,
int &ruleType,
bool &deletionError)

Operation
Returns errors found after a call to spell(Buffer, Object, string, int, int, int, int, int, bool, bool). Note that this perm does not return errors found after calling any other variant of the spell perm.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returnBuffer</td>
<td>If an error is found, the description will be placed in this buffer - must be created before calling.</td>
</tr>
<tr>
<td>wordStart</td>
<td>If an error is found, returns the start position of the incorrect word in the attribute text.</td>
</tr>
<tr>
<td>wordEnd</td>
<td>If an error is found, returns the end position of the incorrect word.</td>
</tr>
<tr>
<td>sentenceStart</td>
<td>If an error is found, returns the start position of the sentence containing the error.</td>
</tr>
<tr>
<td>sentenceEnd</td>
<td>If an error is found, returns the end position of the sentence containing the error.</td>
</tr>
<tr>
<td>correctionComplete</td>
<td>This flag will be returned true if no more errors were found.</td>
</tr>
<tr>
<td>skipSentence</td>
<td>Set this flag when calling to ignore any remaining errors in the current sentence.</td>
</tr>
<tr>
<td>ruleType</td>
<td>If an error is found, returns the code of the rule that triggered the error.</td>
</tr>
<tr>
<td>deletionError</td>
<td>If an error is found, this flag indicates that the error type recommends that text is deleted.</td>
</tr>
</tbody>
</table>

SpellingErrors__

Declaration
SpellingErrors__ spellingErrors()

Operation
Structure encapsulating information about spelling and grammatical errors.

for SpellingError in SpellingErrors__

Declaration
for SpellingError in SpellingErrors__
Operation
Loop to iterate over errors found after calling the `spell(Buffer, Object, string, int&, int&, int&, int&, int&, bool, bool&)` perm. Note that this loop does not list errors found after calling any other variant of the spell perm.

getErrorString

Declaration
Buffer getErrorString(Buffer returnBuffer, SpellingError spellErr)

Operation
Returns a description for the specified error `spellErr`. The `returnBuffer` parameter must be created before calling.

gerErrorStartPos(SpellingError)

Declaration
int getErrorStartPos(SpellingError spellErr)

Operation
Used inside the 'for error in spellingErrors do' loop. Returns the position of the start of the spelling/grammatical error relative to the start of the object.

gerErrorStopPos(SpellingError)

Declaration
int getErrorStopPos(SpellingError spellErr)

Operation
Used inside the 'for error in spellingErrors do' loop. Returns the position of the last character of the spelling/grammatical error relative to the start of the object.

gerSentenceStartPos(SpellingError)

Declaration
int getSentenceStartPos(SpellingError spellErr)

Operation
Used inside the 'for error in spellingErrors do' loop. Returns the position of the first character in the sentence containing the spelling/grammatical error relative to the start of the object.
getSentenceStopPos(SpellingError)

Declaration

int getSentenceStopPos(SpellingError spellErr)

Operation

Used inside the 'for error in spellingErrors do' loop. Returns the position of the last character in the sentence containing the spelling/grammatical error relative to the start of the object.

getCorrectionComplete(SpellingError)

Declaration

bool getCorrectionComplete(SpellingError spellErr)

Operation

Used inside the 'for error in spellingErrors do' loop. Returns true if the spell check is complete otherwise false.

ignoreWord

Declaration

void ignoreWord(string word)

Operation

Causes the specified word to be ignored if it is found to be incorrect during spell checking. The word is ignored until resetSpellingState is called.

for Buffer in SpellingAlternatives__

Declaration

for b in spellAlt do {
  ...
}

where:

b is a variable of type Buffer

spellAlt is a variable of type SpellingAlternatives__
Operation
A loop to iterate through alternative words found after a spelling error. Alternative words are returned in a Buffer object, but note that the user should not create or destroy the Buffer.

alternative

Declaration
Buffer alternative(Buffer returnBuffer, int index)

Operation
Returns the alternative word at the specified index position, after a spelling error. The returnBuffer parameter must be created before calling. An error will be reported if the index is out of range.

Language and Grammar

Languages

Declaration
Languages__ languages()

Operation
Type to iterate through spLanguageInfo

Language

Declaration
Language lang

Operation
Type to encapsulate details of available language databases.

spGetLanguages

Declaration
int spGetLanguages()
Operation
Fills the list of available languages. This will be a list of those languages that are supported by the spell checker and whose language database is present on the local client. Returns the number of available languages.

for Language in Languages__

Declaration
for Language in Languages__
Operation
Iterates through the specified Languages.

getLanguage

Declaration
Language getLanguage(int index)
Operation
Returns the spLanguageInfo structure for the language at the specified index in the list of available languages. If the index value is outwith the range of available languages, an error report is generated.

getId

Declaration
int getId(Language lang)
Operation
Returns the ID of the specified language lang. (e.g. "English", "German").

getName

Declaration
string getName(Language lang)
Operation
Returns a string identifying the specified language lang. (e.g. "UK English", "German").
**isSupported**

Declaration

```dxl
bool isSupported(Language lang)
```

Operation

Returns a boolean indicating if the specified language `lang` is officially supported by Rational DOORS. At present, this covers US English, UK English, French and German (pre- and post-Reform).

**getGrammarRules**

Declaration

```dxl
int getGrammarRules(SpellOptions &spellOptions, GrammarRules &gramRules)
```

Operation

Gets a list of active grammar rules for the specified options set `spellOpt`, returning the number of active rules `&gramRules`. Active rules are determined by the current language and grammar strictness level.

The `GrammarRules` parameter must be initialized to 'null' before calling.

**getName**

Declaration

```dxl
Buffer getName(GrammarRules &gramRules, int index, Buffer buf)
```

Operation

Returns the short name of the grammar rule at the position specified by the `index` parameter in the set of grammar rules `&gramRules`. If the value of `index` is greater than the number of active rules, an error report is generated. The `buf` parameter is a Buffer that is used to create the return value; it must be created by the user before calling and deleted afterwards.

**getExplanation**

Declaration

```dxl
string getExplanation(GrammarRules &gramRules, int index, Buffer buf)
```

Operation

Returns a full explanation of the grammar rule at the position specified by the `index` parameter in the set of grammar rules `&gramRules`. If the value of `index` is greater than the number of active rules, an error report is generated. The `buf` parameter is a Buffer that is used to create the return value; it must be created by the user before calling and deleted afterwards.
**getOptions**

Structure encapsulating spell checker options.

Declration

```c
string getOptions(SpellingOptions &spellOptions, int optionsSet)
```

Operation

Gets set of spelling options. The `optionsSet` parameter indicates which set of options to load. At present this is limited to the database-wide default settings (defined by `spDatabaseOptions` with a value of 0) and the user's personal setting (`spUserOptions` with a value of 1). Any other value will return a failure message. Where a user’s settings have not yet been configured, the database default values will be returned. An error string is returned if there is a problem reading the options files, but in this case the `SpellingOptions` parameter will contain standard defaults.

**save(SpellingOptions)**

Declration

```c
string save(SpellingOptions &spellOptions)
```

Operation

Saves the spelling options.

If the options was loaded as the database defaults, the user must have sufficient access rights to modify database settings, otherwise the function will return an error string.

**getLanguage**

Declration

```c
int getLanguage(SpellingOptions &spellOptions)
```

Operation

Returns the ID of the spelling checking language defined in the `SpellingOptions` parameter.

**setLanguage**

Declration

```c
string setLanguage(SpellingOptions &spellOptions, int languageId)
```

Operation

Sets the spell checking language in the specified set of `SpellingOptions`. If the `languageId` is invalid, the function will return an error string.
getEnglishOptions

**Declaration**

```c
string getEnglishOptions(SpellingOptions &spellOptions, bool &legalLexicon, bool &financialLexicon)
```

**Operation**

Returns boolean values indicating the state of options specific to US and UK English. Returns error string if the parameters are missing from the options set.

setEnglishOptions

**Declaration**

```c
string setEnglishOptions(SpellingOptions &spellOptions, bool &legalLexicon, bool &financialLexicon)
```

**Operation**

Sets a boolean value indicating the state of options specific to US and UK English. These can be modified if a language other than English is selected.

getUKOptions

**Declaration**

```c
string getUKOptions(SpellingOptions &spellOptions, bool &izeEndings)
```

**Operation**

Returns a boolean value indicating the state of an option specific to UK English.

setUKOptions

**Declaration**

```c
string setUKOptions(SpellingOptions spellOptions, bool izeEndings)
```

**Operation**

Sets a boolean value indicating the state of an option specific to UK English. This option can be set even if a language other than UK English is selected.
getFrenchOptions

Declaration

string getFrenchOptions(SpellingOptions &spellOptions, bool &openLigature, bool &accentedUpperCase)

Operation

Gets boolean values indicating the state of options specific to French.

setFrenchOptions

Declaration

string setFrenchOptions(SpellingOptions &spellOptions, bool &openLigature, bool &accentedUpperCase)

Operation

Sets boolean values indicating the state of options specific to French. These options can be set even if a language other than French is selected.

getGermanOptions

Declaration

string getGermanOptions(SpellingOptions &spellOptions, bool &scharfes)

Operation

Gets boolean values indicating the state of options specific to German.

setGermanOptions

Declaration

string setGermanOptions(SpellingOptions &spellOptions, bool &sharfes)

Operation

Sets boolean values indicating the state of options specific to German. These options can be set even if a language other than German is selected.
### getGreekOptions

**Declaration**

```c
string getGreekOptions(SpellingOptions, bool& accentedUpperCase)
```

**Operation**

Gets boolean values indicating the state of options specific to Greek.

### setGreekOptions

**Declaration**

```c
string setGreekOptions(SpellingOptions, bool accentedUpperCase)
```

**Operation**

Sets boolean values indicating the state of options specific to Greek. These options can be set even if a language other than Greek is selected.

### getSpanishOptions

**Declaration**

```c
string getSpanishOptions(SpellingOptions, bool& accentedUpperCase)
```

**Operation**

Gets boolean values indicating the state of options specific to Spanish.

### setSpanishOptions

**Declaration**

```c
string setSpanishOptions(SpellingOptions, bool accentedUpperCase)
```

**Operation**

Sets boolean values indicating the state of options specific to Spanish. These options can be set even if a language other than Spanish is selected.

### getCatalanOptions

**Declaration**

```c
string getCatalanOptions(SpellingOptions, bool& periodMode)
```
Operation
Gets boolean values indicating the state of options specific to Catalan.

setCatalanOptions

Declaration
string setCatalanOptions(SpellingOptions, bool periodMode)

Operation
Sets boolean values indicating the state of options specific to Catalan. These options can be set even if a language other than Catalan is selected.

getRussianOptions

Declaration
string getRussianOptions(SpellingOptions, bool& joMode)

Operation
Gets boolean values indicating the state of options specific to Russian.

setRussianOptions

Declaration
string setRussianOptions(SpellingOptions, bool joMode)

Operation
Sets boolean values indicating the state of options specific to Russian. These options can be set even if a language other than Russian is selected.

getGrammarLevel

Declaration
int getGrammarLevel(SpellingOptions &spellOptions)

Operation
Returns an integer value indicating the strictness of grammar checking.
setGrammarLevel

**Declaration**

```cpp
string setGrammarLevel(SpellingOptions &spellOptions, int grammar)
```

**Operation**

Sets an integer value indicating the strictness of grammar checking. Returns an error string if the grammar level is invalid, or if the user does not have sufficient rights to modify settings.

setSpellingCheckingMode

**Declaration**

```cpp
string setSpellingCheckingMode(int spellMode)
```

**Operation**

Sets the mode for spell checking - the parameter is a value indicating spelling only, quick proof, or full proof modes. Returns an error string if the mode value is invalid or the user does not have sufficient rights to modify settings.

getSpellingCheckingMode

**Declaration**

```cpp
int getSpellingCheckingMode()
```

**Operation**

Returns a value indicating the current spell checking mode.

getSpellingFirst

**Declaration**

```cpp
bool getSpellingFirst(SpellOptions &spellOptions)
```

**Operation**

Returns a flag indicating if spelling errors are to be returned before grammar errors in the specified options set.

setSpellingFirst

**Declaration**

```cpp
string setSpellingFirst(SpellOptions &spellOptions, bool errors)
```
Operation
Sets a flag indicating if spelling errors are to be returned before grammar errors in the specified options set. Returns an error string if the user does not have sufficient rights to modify settings.

getIgnoreReadOnly

Declaration
bool getIgnoreReadOnly(SpellOptions &spellOptions)

Operation
Returns a flag indicating if objects that are read only are to be ignored (not checked) in the specified options set.

setIgnoreReadOnly

Declaration
string setIgnoreReadOnly(SpellOptions &spellOptions, bool read)

Operation
Sets a flag indicating if objects that are read only are to be ignored (not checked) in the specified options set. Returns an error string if the user does not have sufficient rights to modify settings.

Spelling Dictionary

Dictionary
A new type to represent a dictionary, including its type (database or client) and contents. A variable of this type should be initialized to null before opening the dictionary.

open(Dictionary)

Declaration
string open(Dictionary &dict, int dictionaryType)
string open(Dictionary &dict, int languageId, int dictionaryType)

Operation
The first form opens a client or database dictionary for the language defined in the current user’s spelling options. The Dictionary parameter should be initialized to null before calling this function. Returns a string indicating failure, or null if successful. A dictionary must be opened to make its contents available to the spell checker.
Note that there is an upper limit on the number of dictionaries that can be opened at any one time, so it is important that the dictionary is explicitly closed using `spCloseDictionary` after use.

This function will load the dictionary ACL if the dictionary type is set to `spDatabaseDictionary`.

The second form opens a client or database dictionary for the language specified. This opens a temporary dictionary for management functions (such as adding and removing words), and the contents of this dictionary will not be used in normal spell checking. The Dictionary parameter should be initialized to null before calling this function. Returns a string indicating failure, or null if successful.

It is important that the dictionary is explicitly closed using `spCloseDictionary` after use.

This function will load the dictionary ACL if the dictionary type is set to `spDatabaseDictionary`.

---

**close(Dictionary)**

**Declaration**

```dxl
string close(Dictionary &dict, bool saveContents)
string close(Dictionary &dict, bool saveContents, bool saveACL)
```

**Operation**

The first form closes the specified dictionary. If the `saveContents` parameter is `true`, and the user has sufficient permissions, the contents of the dictionary will be saved. This function will not save any changes to the dictionary access control list. Note that this function resets the dictionary parameter to 'null'.

The second form closes the specified dictionary. If the `saveContents` parameter is `true`, and the user has sufficient permissions, the contents of the dictionary will be saved. If the `saveACL` parameter is `true`, and the dictionary type was `Database` dictionary, and the user has sufficient permissions, the dictionary access control list will be saved. Note that this function resets the dictionary parameter to 'null'.

---

**alternativeWord**

**Declaration**

```dxl
alternativeWord
```

**Operation**

Structure to encapsulate a word and its suggested alternative.

---

**for Buffer in Dictionary**

**Declaration**

```dxl
for b in &dict do {
...
}
```
where:

- $b$ is a variable of type `Buffer`
- `&dict` is a variable of type `Dictionary`

**Operation**

Iterator over the words in a dictionary. The user does not need to create the `Buffer` before the loop; the user should not delete the `Buffer` inside or after the loop.

```dxl
for alternativeWord in Dictionary
```

**Declaration**

```dxl
for altWord in &dict do {
    ...
}
```

where:

- `altWord` is a variable of type `alternativeWord`
- `&dict` is a variable of type `Dictionary`

**Operation**

Iterator over the alternative words in a dictionary.

**getWord**

**Declaration**

```dxl
Buffer getWord(alternativeWord altWord, Buffer b)
```

**Operation**

Returns the word component of an `spAltWord` structure. The `b` parameter is used to create the return value and should be created before calling and deleted afterwards.

**getAlternative**

**Declaration**

```dxl
Buffer getAlternative(alternativeWord altWord, Buffer b)
```
Operation
Returns the alternative word component of an spAltWord structure. The b parameter is used to create the return value and should be created before calling and deleted afterwards.

insert

Declaration
string insert(Dictionary &dict, string word, string alternative)

Operation
Adds a word and a preferred alternative word to the specified dictionary.
Returns an error string if the user does not have sufficient rights to modify the dictionary.

remove

Declaration
string remove(Dictionary &dict, string word)

Operation
Removes a word from the specified dictionary.
Returns an error string if the user does not have sufficient rights to modify the dictionary.

isDatabaseDict

Declaration
bool isDatabaseDict(Dictionary &dict)

Operation
Returns a flag indicating whether the specified dictionary is a database dictionary (true) or client dictionary (false).

Miscellaneous Spelling

anagram

Declaration
bool anagram(string word, int minLength)
Operation

Gets up to a maximum of twenty anagrams of the specified word, with the specified minimum length.

Returns a flag indicating if any anagrams were found.

Anagrams are accessed by the same method as getting spelling alternatives.

Example

```dxl
if (anagram("word", 2))
{
    string s
    for s in alternatives do
    {
        print s "\n"
    }
}
```

---

wildcard

Declaration

```dxl
bool wildcard(string pattern)
```

Gets up to a maximum of twenty wildcards based on the specified pattern. The pattern string, a ’?’ matches a single letter, and "*" matches zero or more letters.

Returns a flag indicating if any wildcard matches were found.

Wildcard matches are accessed by the same method as getting spelling alternatives.

Example

```dxl
if (wildcard("w?a*d"))
{
    string s
    for s in alternatives do
    {
        print s "\n"
    }
}
```
Spelling\Dictionary Examples

Example 1
//Check single word and show corrections

class SpellCorrectors

string result
result = spell("helo")
if (!null result)
{
    print result "\n"
    Buffer suggestion
    for suggestion in alternatives do
    {
        print stringOf(suggestion) "\n"
    }
}

Example 2
//Open dictionary and show contents

class SpellCorrectors

Dictionary dict
if (null open(dict, databaseDictionary))
{
    print "Words\n"
    Buffer word
    for word in dict do
    {
        print stringOf(word) "\n"
    }
}

print "\n"
AlternativeWord altWord
Buffer wordBuffer = create
for altWord in dict do
{
    print stringOf(getWord(altWord, wordBuffer) )
    print " -> "
    print stringOf(getAlternative(altWord, wordBuffer)) "\n"
}
delete wordBuffer
print close(dict, true)

Example 3
//List names of available languages

Language language
if (getLanguages > 0)
{
    for language in languages do
    {
        print getName(language) "\n"
    }
}

Example 4
//Open user's spell settings and show current language
SpellingOptions options
getOptions(options, userSpellingOptions)
// get the user's current language
int languageId = getLanguage(options)
// get the details for this language
Language language = getLanguage(languageId)

print getName(language) "\n"
// set the language to French and save the options
print setLanguage(options, French)
print saveOptions(options)
Example 5

//Show grammar options and active rules
SpellingOptions options
getOptions(options, userSpellingOptions)
int grammarLevel = getGrammarLevel(options)
int checkingMode = getCheckMode(options)

if (grammarLevel == formalGrammar)
    print "Formal \n"
else if (grammarLevel == standardGrammar)
    print "Standard \n"
else
    print "Informal \n"
if (checkingMode == spellingOnly)
    print "Spelling Only\n"
else if (checkingMode == quickProof)
    print "Quick Proof \n"
else
    print "Full Proof \n"
GrammarRules rules = null
int numRules = getGrammarRules(options, rules)
int index
Buffer ruleName = create
for (index = 0; index < numRules; index++)
{
    print stringOf(getName(rules, index, ruleName)) "\n"
}
delete ruleName
Chapter 34
Database Integrity Checker

This chapter describes the database integrity checker.

- Database Integrity Types
- Database Integrity Perms

Database Integrity Types

IntegrityResultsData

This type is a handle to an object that is created and returned by the `checkDatabaseIntegrity` perm (see below), and which contains the results of the integrity check.

IntegrityCheckItem

This type is contained in an ordered list in the IntegrityResultsData object. Each item in the list corresponds to the start or completion of the checking of a folder, or a discovered inconsistency (problem) with the data integrity.

IntegrityProblem

This type is contained in a list in the IntegrityResultsData object. Each item corresponds to a problematical reference to a hierarchy Item by one folder in the hierarchy, or, in the case of orphaned items, to an item that is not referenced by any folder.

ProblemItem

This type is contained in a list in the IntegrityResultsData object. Each item corresponds to a hierarchy Item that has one or more IntegrityProblem records associated with it.

IntegrityItemType

This enumerated type is returned by the `type(IntegrityCheckItem)` and `type(IntegrityProblem)` perms, to identify the meaning of each item.

Both perms can return the following values:
• referencesInvalidFolder
• referencesValidFolder
• noDataFound
• orphanedItem
• invalidProjectListEntry
• missingProjectListEntry

In addition, the type(IntegrityCheckItem) perm can return the following values:
• startedCheck
• completedCheck
• failedCheck

---

**Database Integrity Perms**

The Database Integrity functionality is only accessible to administrator users. For any other user, the checkDatabaseIntegrity perm (see below) returns null, without performing any database integrity checks.

In general, the following perms generate run-time DXL errors when passed null arguments.

---

**checkDatabaseIntegrity(Folder&, IntegrityResultsData&)**

**Declaration**

```dxl
string checkDatabaseIntegrity(Folder& orphansFolder, IntegrityResultsData& results)
```

**Operation**

Performs an integrity check on the database, and returns the results in an IntegrityResultsData object.

Parent/child references are checked for consistency, the database project list is checked for missing items, and the database file system is scanned for orphaned items (data that has become detached from the folder hierarchy, and is therefore no longer accessible to Rational DOORS clients), and these items are placed in the specified orphans folder.

The perm returns null on success, and an error string on failure.

Passing a null argument to this perm causes a run-time error.

Note that the value of the Folder argument may be changed by the checking process.

For any user other than administrator, this perm will not perform any checking, and will return an error string and set results to null.
checkFolderIntegrity(Folder, IntegrityResultsData&, bool)

Declaration

string checkFolderIntegrity(Folder f, IntegrityResultsData& results, bool recurse)

Operation

Performs a parent/child reference consistency check on the contents of the specified folder, checks for a missing entry in the global project list if the folder is a Project, and puts the results in the IntegrityResultsData object. If the recurse argument is true, it performs the same check on all descendants of the folder.

This perm is restricted to administrator users. Error conditions are handled as by the checkDatabaseIntegrity perm.

canceled/cancelled(IntegrityResultsData)

Declaration

bool canceled/cancelled(IntegrityResultsData results)

Operation

Returns true if the integrity check was cancelled by the user pressing the cancel button on the progress bar.

for IntegrityCheckItem in IntegrityResultsData

Declaration

for integchkitem in integresdata

Operation

This iterator returns the IntegrityCheckItem objects in the order in which they were created during the integrity check. Information from these objects can then be used to compile a report of the integrity check.

for ProblemItem in IntegrityResultsData

Declaration

for probitem in integresdata

Operation

This iterator returns an object for each hierarchy item probitem for which one or more problems are found.
for IntegrityProblem in ProblemItem

Declaration

\texttt{for integprob in probitem}

Operation

This iterator returns an object for each problem found for the same item.

for IntegrityProblem in IntegrityResultsData

Declaration

\texttt{for integprob in integresdata}

Operation

This returns all \texttt{IntegrityProblem} objects in the \texttt{IntegrityResultsData} object. These are grouped by unique ID.

\textbf{uniqueID(IntegrityCheckItem)}

Declaration

\texttt{string uniqueID(IntegrityCheckItem integchkitem)}

Operation

This returns the index of the item to which the \texttt{IntegrityCheckItem} applies. For Started/Failed/CompletedCheck items, it refers to the folder whose contents are being checked. For others, it refers to the item in the folder that exhibits a problem.

\textbf{uniqueID(IntegrityProblem)}

Declaration

\texttt{string uniqueID(IntegrityProblem integprob)}

Operation

This returns the index of the item that exhibits the problem.

\textbf{uniqueID(ProblemItem)}

Declaration

\texttt{string uniqueID(ProblemItem probitem)}
Operation
This returns the index of the problem item probitem.

problems(IntegrityResultsData, string)

Declaration
ProblemItem problems(IntegrityResultsData integresdata, string uniqueID)

Operation
This is the converse of the unique ID perm above, returning the ProblemItem for a given index. It returns null if no problems are listed in the IntegrityResultsData variable for the given Item index.

timestamp(IntegrityCheckItem)

Declaration
Date timestamp(IntegrityCheckItem integchkitem)

Operation
This returns the timestamp (date and time) indicating when the IntegrityCheckItem was generated.

folder(IntegrityProblem)

Declaration
Folder folder(IntegrityProblem integprob)

Operation
Returns the parent folder that contains the problematical reference. It returns null for orphaned items.

type(IntegrityCheckItem)

Declaration
IntegrityItemType type(IntegrityCheckItem integchkitem)

Operation
Can return any one of the following values:
• referencesInvalidFolder
• referencesValidFolder
• noDataFound
• orphanedItem
• invalidProjectListEntry
• missingProjectListEntry
• startedCheck
• completedCheck
• failedCheck

type(IntegrityProblem)

Declaration
IntegrityItemType type(IntegrityProblem integprob)

Operation
Can return any one of the following values:
• referencesInvalidFolder
• referencesValidFolder
• noDataFound
• orphanedItem.
• invalidProjectListEntry
• missingProjectListEntry

type(ProblemItem)

Declaration
IntegrityItemType type(ProblemItem probitem)

Operation
Returns the type of the first problem associated with the specified item probitem. It can return any one of the following values:
• referencesInvalidFolder
• referencesValidFolder
• noDataFound
• orphanedItem
• invalidProjectListEntry
• missingProjectListEntry
text(IntegrityCheckItem)

Declaration

string text(IntegrityCheckItem integchkitem)

Operation

Returns the error message string (if any) for the given IntegrityCheckItem.

parentRefID(IntegrityProblem | ProblemItem)

Declaration

string parentRefID(IntegrityProblem integprob| ProblemItem probitem)

Operation

Returns the index of the parent referenced by the problem item’s data.

parentRef(IntegrityProblem | ProblemItem)

Declaration

Folder parentRef(IntegrityProblem integprob| ProblemItem probitem)

Operation

Returns the folder that the problem item references as its parent, if the folder exists.

setParent(ProblemItem, Folder)

Declaration

string setParent(ProblemItem probitem, Folder f)

Operation

Sets the parent of the item referenced by the ProblemItem to the specified folder, removing all references in any other folders that are known to list the item in their contents (other parent folders associated with IntegrityProblems for the same ProblemItem). The perm returns null on success, and an error string on failure. If the specified Folder does not already contain a reference to the item, then a reference is added. The affected IntegrityProblems are marked as repaired (see below).

If the ProblemItem type is noDataFound, then the Folder argument can be null, in which case all folder entries are removed. If the ProblemItem type is noDataFound and the folder argument is not null, new data is created if the item is a Project or Folder.
addProjectEntry(ProblemItem)

Declaration

string addProjectEntry(ProblemItem probitem)

Operation

If the ProblemItem includes a "missingProjectListEntry" problem, this perm adds the missing entry in the global project list. On successful completion, it marks any missingProjectListEntry problems for the ProblemItem as repaired, and returns null. On failure, it returns an error message.

If the entry name matches an existing entry (project), or a top-level folder in the database, then the project is renamed by appending a space and a decimal integer (defaulting to 1, but incremented as required to achieve uniqueness).

convertToFolder(ProblemItem)

Declaration

string convertToFolder(ProblemItem probitem)

Operation

If the ProblemItem includes a missingProjectListEntry problem, this perm converts the referenced Project to a Folder. In the case of duplicate references to the item, this is done for all known references. On successful completion, the perm marks the affected IntegrityProblems as repaired, and returns null. On failure, it returns an error message.

repaired(IntegrityProblem)

Declaration

bool repaired(IntegrityProblem integprob)

Operation

Returns true if the problem described by the specified IntegrityProblem has been repaired by the setParent perm.

repaired(ProblemItem)

Declaration

bool repaired(ProblemItem probitem)

Operation

Returns true if all of the problems associated with the specified IntegrityProblem have been repaired by the setParent perm.
**delete(IntegrityResultsData&)**

**Declaration**

```plaintext
string delete(IntegrityResultsData& integresdata)
```

**Operation**

Deletes the IntegrityResultsData object and sets its value to null.

---

**checkItem(IntegrityProblem)**

**Declaration**

```plaintext
IntegrityCheckItem checkItem(IntegrityProblem integprob)
```

**Operation**

Returns an IntegrityCheckItem reference, for the given IntegrityProblem. The returned value can be passed to any perm or function taking an IntegrityCheckItem argument.

---

**everSectioned**

**Declaration**

```plaintext
bool everSectioned(ModName_|ModuleVersion)
```

**Operation**

Reports whether the specified ModName_ or ModuleVersion shows evidence that the module was ever save with shareable sections. If a supplied ModuleVersion references a baseline, rather than a current version, false will be returned.
Chapter 35

Discussions

This chapter describes features that operate on Rational DOORS discussions:

- Discussion Types
- Properties
- Iterators
- Operations
- Triggers
- Discussions access controls
- Example

Discussion Types

Discussion

Represents a discussion.

Comment

Represents a comment in a discussion.

DiscussionStatus

Represents the status of a discussion. The possible values are Open and Closed.

Properties

The following tables describe the properties available for the Discussion and Comment types. Property values can be accessed using the . (dot) operator, as shown in the following syntax:

variable.property
where:

- **variable** is a variable of type `Discussion` or `Comment`
- **property** is one of the discussion or comment properties

### Discussion

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>DiscussionStatus</td>
<td>The status of the discussion: whether it is open or closed.</td>
</tr>
<tr>
<td>summary</td>
<td>string</td>
<td>The summary text of the discussion, which may be null</td>
</tr>
<tr>
<td>createdBy</td>
<td>User</td>
<td>The user who created the discussion, if it was created in the current database. Otherwise it returns null.</td>
</tr>
<tr>
<td>createdByFullName</td>
<td>string</td>
<td>The name of the user who created the discussion, as it was when the discussion was created.</td>
</tr>
<tr>
<td>createdByByName</td>
<td>string</td>
<td>The full name of the user who created the discussion, as it was when the discussion was created.</td>
</tr>
<tr>
<td>createdOn</td>
<td>Date</td>
<td>The date and time the discussion was created.</td>
</tr>
<tr>
<td>createdDataTimestamp</td>
<td>Date</td>
<td>The last modification timestamp of the object or module that the first comment in the discussion referred to.</td>
</tr>
<tr>
<td>lastModifiedBy</td>
<td>User</td>
<td>The user who added the last comment to the discussion, or who last changed the discussion status.</td>
</tr>
<tr>
<td>lastModifiedByName</td>
<td>string</td>
<td>The user name of the user who added the last comment to the discussion, or who last changed the discussion status.</td>
</tr>
<tr>
<td>lastModifiedByFullName</td>
<td>string</td>
<td>The full name of the user who added the last comment to the discussion, or who last changed the discussion status.</td>
</tr>
<tr>
<td>lastModifiedDataTimestamp</td>
<td>Date</td>
<td>The last modification timestamp of the object or module that the last comment in the discussion referred to.</td>
</tr>
<tr>
<td>lastModifiedOn</td>
<td>Date</td>
<td>The date and time the last comment was added, or when the discussion status was last changed.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Extracts</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>firstVersion</td>
<td>ModuleVersion</td>
<td>The version of the module the first comment was raised against.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> If a comment is made against the current version of a module and the module is then baselined, this property will return a reference to that baseline. If the baseline is deleted, it will return the deleted baseline.</td>
</tr>
<tr>
<td>lastVersion</td>
<td>ModuleVersion</td>
<td>The version of the module the latest comment was raised against. See note for the <code>firstVersion</code> property above.</td>
</tr>
<tr>
<td>firstVersionIndex</td>
<td>string</td>
<td>The baseline index of the first module version commented on in the discussion. Can be used in comparisons between module versions.</td>
</tr>
<tr>
<td>lastVersionIndex</td>
<td>string</td>
<td>The baseline index of the last module version commented on in the discussion. Can be used in comparison between module versions.</td>
</tr>
</tbody>
</table>

**Comment**

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>string</td>
<td>The plain text of the comment.</td>
</tr>
<tr>
<td>moduleVersionIndex</td>
<td>string</td>
<td>The baseline index of the module version against which the comment was raised. Can be used in comparisons between module versions.</td>
</tr>
<tr>
<td>status</td>
<td>DiscussionStatus</td>
<td>The status of the discussion in which the comment was made.</td>
</tr>
<tr>
<td>moduleVersion</td>
<td>ModuleVersion</td>
<td>The version of the module against which the comment was raised.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> If a comment if made against the current version of a module and the module is then baselined, this property will return a reference to that baseline. If the baseline is deleted, it will return the deleted baseline.</td>
</tr>
<tr>
<td>onCurrentVersion</td>
<td>bool</td>
<td>True if the comment was raised against the current version of the module or an object in the current version.</td>
</tr>
</tbody>
</table>
### Property | Type | Extracts
--- | --- | ---
changedStatus | bool | Tells whether the comment changed the status of the discussion when it was submitted. This will be true for comments which closed or re-opened a discussion.
dataTimestamp | Date | The last modified time of the object or module under discussion, as seen at the commenting user's client at the time the comment was submitted.
createdBy | User | The user that created the comment. Returns null if the user is not in the current user list.
createdByName | string | The user name of the user who created the comment, as it was when the comment was created.
createdByFullName | string | The full name of the user who created the comment, as it was when the comment was created.
createdOn | Date | The date and time when the comment was created.
discussion | Discussion | The discussion containing the comment.

## Iterators

### for Discussion in Type

**Syntax**

```dxl
for disc in Type do {
 ...
}
```

where:

- `disc` is a variable of type `Discussion`
- `Type` is a variable of type `Object, Module, Project or Folder`

**Operation**

Assigns the variable `disc` to be each successive discussion in `Type` in the order they were created. The first time it is run the discussion data will be loaded from the database.

The `Module`, `Folder` and `Project` variants will not include discussions on individual objects.
Note: The Folder and Project variants are provided for forward compatibility with the possible future inclusion of discussions on folders and projects.

for Comment in Discussion

Syntax

for comm in disc do {
  ...
}

where:

comm is a variable of type Comment
disc is a variable of type Discussion

Operation

Assigns the variable comm to be each successive comment in disc in chronological order. The first time it is run on a discussion in memory, the comments will be loaded from the database. Note that if a discussion has been changed by a refresh (e.g. in terms of the last Comment timestamp) then this will also refresh the comments list.

The discussion properties will be updated in memory if necessary, to be consistent with the updated list of comments.

Operations

create(Discussion)

Declaration

string create(target, string text, string summary, Discussion& disc)

Operation

Creates a new Discussion about target, which can be of type Object or Module. Returns null on success, error string on failure. Also add text as the first comment to the discussion.

addComment

Declaration

string addComment(Discussion disc, target, string text, Comment& comm)
Operation

Adds a Comment about target to an open Discussion. Note that target must be an Object or Module that the Discussion already relates to. Returns null on success, error string on failure.

closeDiscussion

Declaration

string closeDiscussion(Discussion disc, target, string text, Comment& comm)

Operation

Closes an open Discussion disc by appending a closing comment, specified in text. Note that target must be an Object or Module that disc already relates to. Returns null on success, error string on failure.

reopenDiscussion

Declaration

string reopenDiscussion(Discussion disc, target, string text, Comment& comm)

Operation

Reopens a closed Discussion disc and appends a new comment, specified in text. Note that target must be an Object or Module that disc already relates to. Returns null on success, error string on failure.

deleteDiscussion

Declaration

string deleteDiscussion(Discussion d, Module m|Object o)

Operation

Deletes the specified module or object discussion if the user has the permission to do so. Returns null on success, or an error string on failure.

sortDiscussions

Declaration

void sortDiscussions({Module m|Object o|Project p|Folder f}, property, bool ascending)

Operation

Sorts the discussions list associated with the specified item according to the given property, which may be a date, or a string property as listed in the discussions properties list. String sorting is performed according to the lexical ordering for the current user's default locale at the time of execution.
If the discussion list for the specified item has not been loaded from the database, this perm will cause it to be loaded.

**Note:** The Folder and Project forms are provided for forward compatibility with the possible future inclusion of discussions on folders and projects.

---

### getDiscussions

**Declaration**

```dxl
cstring getDiscussions({Module m|Object o|Project p|Folder f})
```

**Operation**

Refreshes from the database the Discussion data for the specified item in memory. Returns null on success, or an error on failure.

---

### getObjectDiscussions

**Declaration**

```dxl
cstring getObjectDiscussions(Module m)
```

**Operation**

Refreshes from the database all Discussions for all objects in the specified module. Returns null on success, or an error on failure.

---

### getComments

**Declaration**

```dxl
cstring getComments(Discussion d)
```

**Operation**

Refreshes from the database the comments data for the specified Discussion in memory. Returns null on success, or an error on failure.

**Note:** The Discussion properties will be updated if necessary, to be consistent with the updated comments list.

---

### mayModifyDiscussionStatus

**Declaration**

```dxl
cbool mayModifyDiscussionStatus(Discussion d, Module m)
```

**Operation**

Checks whether the current user has rights to close or re-open the specified discussion on the specified module.
baselineIndex

Declaration

string baselineIndex(Module m)

Operation

Returns the baseline index of the specified Module, which may be a baseline or a current version. Can be used to tell whether a Comment can be raised against the given Module data in a given Discussion.

Note: A Comment cannot be raised against a baseline index which is less than the lastVersionIndex property of the Discussion.

isDiscussionColumn

Declaration

bool isDiscussionColumn(Column c)

Operation

Returns true if the column is a discussion column, otherwise false.

setDiscussionColumn

Declaration

void setDiscussionColumn(Column c, string s)

Operation

Sets the filter on the discussion column based on the supplied discussion DXL filename.

Example

Column c
for c in current Module do
{
  if (isDiscussionColumn(c))
  {
    string s = dxlFilename(c)
    if (s != null)
    {
      Module m = edit("/TestDiscussions ", true)
      //Open a module, with some discussions in it.
if (m != null)
{
    Column cNew = insert(column 3)
    title(cNew, "My copy Discussion")
    string home = getenv("HOME")
    string fullPath = home "\" s ""
    string contents = readFile(fullPath)
    //Call dxl PERM on that column before setting the discussion column. The discussion column is also a modified version of LAYOUT dxl.
    dxl(cNew, contents)
    setDiscussionColumn(cNew, s)
    width(cNew, 100)
    refresh(m, false)
}

---

**Triggers**

Trigger capabilities have been expanded so that triggers can now be made to fire before or after a Discussion or a Comment is created.

As follows:

<table>
<thead>
<tr>
<th></th>
<th>pre</th>
<th>post</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Discussion</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

**Declaration**

Comment comment(Trigger t)

**Operation**

Returns the Comment with which the supplied Trigger is associated, null if not a Comment trigger.
discussion

Declaration
Discussion discussion(Trigger t)

Operation
Returns the Discussion with which the supplied Trigger is associated, null if not a Discussion trigger.

dispose(Discussion/Comment)

Declaration
void dispose({Discussion& d|Comment& c})

Operation
Disposes of the supplied Comment or Discussion reference freeing the memory it uses.
Can be called as soon as the reference is no longer required.

Note: The disposing will take place at the end of the current context.

Discussions access controls

This section describes functions that report on access rights for discussions.

canModifyDiscussions

Declaration
bool canModifyDiscussions({Module m| Item i| string s}[, {User |string}])

Operation
Returns true if a given user or named user (current user if the parameter is not supplied) is allowed to create a discussion or a comment on a discussion for the given module, item or named module. The use of item is intended for use when the Item represents a module.

canEveryoneModifyDiscussions

Declaration
bool canEveryoneModifyDiscussions({Module m| Item i})
Operation
Returns true if the discussions access list for the given module or item contains the special "Everyone" group.

addUser

Declaration
void addUser(Item i, {User u| string s})

Operation
Adds the user or named user to the Discussion Access List for an Item. The updated list is not saved in the database until saveDiscussionAccessList is called.

addGroup

Declaration
void addGroup(Item i, {Group g| string s})

Operation
Adds the group or named group to the Discussion Access List for an Item. The updated list is not saved in the database until saveDiscussionAccessList is called.

removeUser

Declaration
void RemoveUser(Item i, {User u| string s})

Operation
Remove the user or named user from the Discussion Access List for an Item. The updated list is not saved in the database until saveDiscussionAccessList is called.

removeGroup

Declaration
void removeGroup(Item i, {Group g| string s})

Operation
Remove the group or named group from the Discussion Access List for an Item. The updated list is not saved in the database until saveDiscussionAccessList is called.
saveDiscussionAccessList

**Declaration**

```dxl
string saveDiscussionAccessList(Item i)
```

**Operation**

This perm saves the discussion access list for the given item to the database. This perm is only successful for an administrator or a user with Manage Database privileges. If the call is successful, a null value will be returned, otherwise a string with an error message will be returned.

**Example**

```dxl
// Create a Discussion on the current Module, with one follow-up Comment...
Module m = current
Discussion disc = null
create(m,"This is my first comment.","First summary",disc)
Comment cmt
addComment(disc, m, "This is the second comment.", cmt)

// Display all Discussions on the Module
for disc in m do
{
    print disc.summary " (" disc.status ")\n"
    User u = disc.createdBy
    string s = u.name
    print "Created By: " s "\n"
    print "Created By Name: \" disc.createdByName "\n"
    print "Created On: " stringOf(disc.createdOn) "\n"
    u = disc.lastModifiedBy
    s = u.name
    print "Last Mod By: " s "\n"
    print "Last Mod By Name: \" disc.lastModifiedByName "\n"
    print "Last Mod On " stringOf(disc.lastModifiedOn) "\n"
```
print "First version: " (fullName disc.firstVersion) " [" //-
     (versionString disc.firstVersion) "]\n"
print "Last version: " (fullName disc.lastVersion) " [" //-
     (versionString disc.lastVersion) "]\n"

Comment c
for c in disc do
{
    print "Comment added by " (c.createdByName) " at " //-
    (stringOf(c.createdOn)) ":\n"
    print "Module Version: " (fullName c.moduleVersion) " [" //-
     (versionString c.moduleVersion) "]\n"
    print "Data timestamp: " (stringOf c.dataTimestamp) "\n"
    print "Status: " c.status " (" (c.changedStatus ? "Changed" //-
     : "Unchanged") ")\n"
    print "On current: " c.onCurrentVersion "\n"
    print c.text "\n"
}
}
Chapter 36

General functions

This chapter describes functions that do not belong to any major grouping.

- Error handling
- Archive and restore
- Checksum validation
- Locking
- HTML functions
- HTTP Server
- Asynchronous HTTP requests
- OSLC DXL Services
- Broadcast Messaging
- Converting a symbol character to Unicode
- Timer

Error handling

This section defines functions for handling errors.

When parse time errors occur when running DXL scripts, the #include nesting of files is reported, in addition to the file and line number of the error.

Take two DXL files, c:\temp\a.dx and c:\temp\b.dx:

```
//file a.dx
#include <c:\temp\b.dx>
//file b.dx
while //syntax error
```

Execute the DXL statement:

```
#include <c:\temp\a.dx>
```

This returns:

```
-E- DXL: <c:\temp\b.dx:2> syntax error
Included from:
  <c:\temp\a.dx:1>
```
Notice that the file containing the error is displayed first, followed by a list of ‘included from’ files.

For runtime error reports of DXL scripts, the function backtrace, or callstack, is reported.

Run the following DXL program:

```dxl
void f()
{
   string s
   print s
}

g
```

The result will be:

```
-R-E- DXL: <Line:4> unassigned variable (s)
Backtrace:
   <Line:9>
   <Line:12>
-I- DXL: execution halted
```

Notice that there is a backtrace showing the function call nesting at the time the runtime error occurred.

---

**error**

**Declaration**

```dxl
void error(string message)
```

**Operation**

Terminates the current DXL program, prints the string `message` in the DXL Interaction window’s output pane, and pops up a modal dialog box, which announces the presence of errors.

**Example**

```dxl
error "No links to trace"
```
lastError

Declaration

string lastError()

Operation

Returns the last error as a string. If the noError function has been called, certain key functions do not fail and halt when they discover an error condition. Instead, they set an error message, which can be checked by this function. Calling lastError terminates noError.

Returns null if there are no errors.

This function can be used to turn error box messages back on after the function has been used.

Example

goError
Module m = share("Key data", false)
string openStatus = lastError
if (null openStatus) {
    // we opened the module for full access
} else {
    // some one has the module open for edit
}

noError

Declaration

void noError()

Operation

Switches off DXL run-time errors until lastError is called. Any function that can produce a run-time error is affected. Instead of failing and halting when they discover an error condition, they set an error message, which can be checked by the lastError function.

Calling this function resets the error message to null, so you must store any potential error messages for reuse.

unixerror

Declaration

void unixerror(string message)

Operation

Similar to the error function except that the last known operating system error is printed, as well as the string message.
The name `unixerror` is not well chosen, because the function works correctly on all Rational DOORS platforms. The name is derived from the UNIX `perror` function.

Example

```dxl
Stat s = create "/no-such-file"
if (null s) unixerror "trouble with filename: 
```

**warn**

**Declaration**

```dxl
t void warn(string message)
```

**Operation**

Similar to the `error` function except that the program is not halted.

**dxlHere()**

**Declaration**

```dxl
t string dxlHere()
```

**Operation**

This returns the file and line of DXL code currently being executed. Useful for debugging DXL scripts. It only returns the file information for DXL scripts executed by using the `#include` mechanism.

**Example**

```dxl
print dxlHere() "\n"
```

**Archive and restore**

This section defines properties, constants, and functions for use with Rational DOORS archive and restore. Two main data types are introduced:

- **ArchiveItem** An item in an archive
- **ArchiveData** A list of the contents of an archive

**Archive properties**

Properties are defined for use with the . (dot) operator and ArchiveItem structure to extract information about archives, as shown in the following syntax:
The following tables list the properties and the information they extract.

### String property

<table>
<thead>
<tr>
<th>Property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>archiveItemName</td>
<td>The name of the archive item</td>
</tr>
<tr>
<td>archiveItemDescription</td>
<td>The description of the archive item</td>
</tr>
<tr>
<td>archiveItemType</td>
<td>The type of the archive item</td>
</tr>
</tbody>
</table>

### Boolean property

<table>
<thead>
<tr>
<th>Property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>archiveItemSelected</td>
<td>Whether the item is selected</td>
</tr>
<tr>
<td>archiveItemSoftDeleted</td>
<td>Whether the item is soft deleted</td>
</tr>
</tbody>
</table>

---

## Syntax

```dxl
for archiveItem in archive do {
    ...
}
```

where:

- `archiveItem` Is a variable of type `ArchiveItem`
- `archive` Is a variable of type `ArchiveData`

## Operation

Assigns `archiveItem` to be each successive archive item in `archive`.

## Example

```
ArchiveData archiveData
string message
message = get("c:\project.dpa",archiveData)
ArchiveItem archiveItem
```
for archiveItem in archiveData do {
    if (archiveItem.archiveItemName == "my module") {
        if (archiveItem.archiveItemSelected) {
            deselect(archiveItem)
        }
    }
}

archive(modules and projects)

Declaration

string archive(string projectName,
               string fileName,
               bool span
               [, bool incBackups,
                ArchiveInclusionDescriptor allbaselines/noBaselines
               [, bool serverArchive]])

string archive(ModName_ modRef,
               string fileName,
               bool span)
               [, bool incBackups,
                ArchiveInclusionDescriptor allbaselines/noBaselines
               [, bool serverArchive]])

string archive(string projectName,
               string user,
               string password,
               string fileName,
               bool span)

Operation

The first form creates an archive of the project projectName, and puts it in fileName. The default file type is .dpa.
The second form creates an archive of the module named modRef, and puts it in fileName. The default file type is .dma.
The third form creates an archive of the project projectName, and puts it in fileName. This form is supported only for compatibility with earlier releases.
The optional incBackups parameter can be used to specify whether database backup files are to be included in the archive.
The optional ArchiveInclusionDescriptor parameter can be used to specify whether baselines will be included in the archive.
The optional `serverArchive` parameter can be used to specify whether the archive will be created in the database server-side archive location.

If `span` is `true` and the path specified is on a removable disk, the archive spans multiple disks.

**Example**

This example archives a module:

```dxl
string message = archive(module "Car", "d:\temp\car.dma", false)
if (!null message) {
    ack message
    halt
}
```

This example archives a project:

```dxl
string message = archive ("Car project",
    "a:\car_project.dpa", true)
if (!null message) {
    ack message
    halt
}
```

---

### archive(user list)

**Declaration**

```dxl
string archive(string fileName,
    bool span)
```

**Operation**

Creates an archive of the user list, and puts it in `fileName`. The default file type is `.dua`.

### archiveFiles

**Declaration**

```dxl
string archiveFiles(string fileName,
    string dir,
    bool span)
```

**Operation**

Archives the files recursively from the directory `dir` into the zip file `fileName`. The zip file is compatible with `pkzip`. If `span` is `true` and the path specified is on a removable disk, the archive spans multiple disks.

**Example**

This example archives all the files in a directory:

```dxl
string message
```
message = archiveFiles("d:\temp\temp.zip",
    "d:\temp\archive\", false)

if (!null message) {
    ack message
    halt
}

---

getArchiveType

**Declaration**

```dxl
string getArchiveType(string fileName, 
    ZipType& zip, [bool serverArchive])
```

**Operation**

Returns the type of archive file as one of the following constants:

- `zipNotArchive`: Archive is not a project or module
- `zipProjectArchive`: Project archive
- `zipModuleArchive`: Module archive
- `zipPre3ProjectArchive`: Project archived under a version of Rational DOORS prior to 3.0
- `zipPre3ModuleArchive`: Module archived under a version of Rational DOORS prior to 3.0
- `zipPre5ProjectArchive`: Project archived under a version of Rational DOORS prior to 5.0
- `zipPre5ModuleArchive`: Module archived under a version of Rational DOORS prior to 5.0
- `zipProject5Archive`: Project archived under Rational DOORS 5.x
- `zipModule5Archive`: Module archived under Rational DOORS 5.x
- `zipUserListArchive`: User list archive
- `zipUserList5Archive`: User list archived under Rational DOORS 5.x

**Example**

This example finds out whether a zip file is a project archive:

```dxl
string file = "d:\temp\temp.dpa"
string message
ZipType zip
message = getArchiveType(file, zip)
if (!null message){
    print "Failed: " message "\n"
    halt
}
```
if (zip == zipProjectArchive) {
    print file " is a project archive from the latest DOORS version"
} else if (zip == zipProject5Archive) {
    print file " is a project archive from DOORS version 5"
} else {
    print file " is neither a version 5 nor a version 6 project archive"
}

getModuleDetails

Declaration

string getModuleDetails(string fileName,
    string &moduleName,
    string &projectName,
    [string &databaseID,
    string &databaseName,]
    Date &archiveDate, [bool serverArchive],[ZipType& zt] ()

Operation

Passes back module details from the archive fileName. Module details comprise the module name, the project name from which it originates, and the archive date. If the optional arguments databaseID and databaseName are supplied, the function passes back the database ID and name.

If the archive is not a module archive, the function passes back a null string for any parameter it cannot identify, and sensible results for the rest.

The last two flags are for indicating that the archive file is on the server and an additional pointer to a variable used to return the type of zip to the caller.

If serverArchive flag is set to true and the user does not have permission to read a server archive, an error message will be returned.

If the call fails, the function returns an error message.

Example

This example passes back the details of the module archived in d:\temp\car.dma.

string moduleName
string projectName
Date archiveDate

string mess= getModuleDetails("d:\\temp\\car.dma", modName,
    projName, archiveDate)
if (!null message) {
    ack message
    halt
} else {
    string d = archiveDate ""
    print "The archived file contains the module "
    moduleName "
    print "and was archived from "
    print "the project " projectName " on the " d
    \n"
}

getProjectDetails

Declaration

string
getProjectDetails(string fileName,
    string &projectName,
    string &projectDescription,
    [string &databaseID,
    string &databaseName,]
    Date &archiveDate,[bool serverArchive],[ZipType& zt])

Operation

Gets project details from the archive fileName. Project details are the name, description and archive date of the project
that was archived. If the optional arguments databaseID and databaseName are supplied, the function passes back
the database ID and name in them.

If the archive is not a project archive, the function passes back a null string for any parameter it cannot identify, and sensible
results for the rest.

The last two flags are for indicating that the archive file is on the server and an additional pointer to a variable used to return
the type of zip to the caller.

If serverArchive flag is set to true and the user does not have permission to read a server archive, an error message
will be returned.

If the call fails, the function returns an error message.

Example

This example gets the details of the project archived in a: \car_project.dpa.

string projectName
string projectDescription
Date archiveDate
string databaseId
string databaseName
message=getProjectDetails("a:\\car_project.zip", projectName,
projectDescription, databaseId,
archiveDate)

if (!null message) {
  ack message
  halt
} else {
  string d = archiveDate ""
  print "The archived file contains the project "
  projectName
  print "with the description "
  projectDescription
  print " from the database called "
  databaseName
  print " with database ID" databaseId
  print "on the " d \\
}

getUserlistDetails

Declaration

string getUserlistDetails(string fileName,
                          string &databaseId,
                          string &databaseName,
                          Date &archiveDate)

Operation

Gets user list details from the archive fileName. User list details are the ID and name of the database from which the archive was taken.

If the archive is not a user list archive, the function passes back a null string for any parameter it cannot identify, and sensible results for the rest.

If the call fails, the function returns an error message.

restore(archive)

Declaration

string restore(ArchiveData archive
               [,string archiveName], [bool serverArchive])

Operation

Restores archive to archiveName. If the operation succeeds, returns a null string; otherwise, returns an error message.
For a project archive, if you specify `archiveName`, this must be a non-existent location. The function then creates a project with this name, and restores the contents of the archive but not the project itself, into the new project. If you do not specify `archiveName`, the function uses the name of the archived project, and restores it to the current location.

For a module archive, if you specify `archiveName`, this must be an existing location. The function then creates the module archive in this existing folder or project. If you do not specify `archiveName`, the function restores the module to the current location.

For a user list archive, if you specify `archiveName`, it is ignored.

The `serverArchive` flag is an additional flag indicating that the archive file is on the server.

If `serverArchive` flag is set to true and the user does not have permission to read a server archive, an error message will be returned.

---

**restoreModule**

**Declaration**

```dxl
string restoreModule(string fileName
                          [,string moduleName], [bool serverArchive])
```

**Operation**

Restores a module from the archive file `fileName`. Optionally renames the module to the name `moduleName`.

If you are restoring a module without defining its name, it can only be restored into a project that does not already contain a module of that name.

If you are restoring a module with a defined name, the `moduleName` must be unique within the restored folder.

The flag `serverArchive` indicates that the archive file is on the server.

If `serverArchive` flag is set to true and the user does not have permission to read a server archive, an error message will be returned.

**Example**

This example restores a module from `d:\temp\car.dma`.

```dxl
string message = restore "d:\temp\car.dma"
if (!null message) {
    ack message
    halt
}
```

This example restores a module from `d:\temp\car.dma` to the module Car user reqts 2.

```dxl
string message = restore("d:\temp\car.dma", "Car user reqts 2")
if (!null message) {
    ack message
    halt
}
```
restoreFiles

Declaration

string restoreFiles(string fileName,
                      string destination)

Operation

Restores all the files from the zip file fileName to the specified directory destination.

Example

This example restores all the files from a zip file.

```dxl
string message = restoreFiles("d:\temp\temp.zip",
                            "d:\temp\new\")
if (message !=null){
    ack message
    halt
}
```

restoreProject

Declaration

string restoreProject(string fileName
                      [,string projectName
                       [,string projectDescription]]
                      [,bool serverArchive])

Operation

Restores a project from the archive file fileName, optionally renaming the project to projectName with the description projectDescription.

If you are restoring a project without defining its name, it can only be restored into a database that does not already contain a project of that name.

If you are restoring a project with a defined name and description, the projectName must be unique.

Example

This example restores the project Car project from the file a:\car_project.dpa.

```dxl
string message = restore "a:\\car_project.dpa"
if (!null message) {
    ack message
    halt
}
```
This example restores a project from `a:\car_project.dpa` to the project *Car project 2* with the description *Restored project*.

```dxl
string message = restore("a:\car_project.dpa", "Car project 2", "Restored project")
if (message != null) {
    ack message halt
}
```

**restoreUserlist**

**Declaration**

```dxl
string restoreUserlist(string fileName)
```

**Operation**

Restores the user list from the archive file `fileName`.

**select(archive item)**

**Declaration**

```dxl
bool select(ArchiveItem item)
```

**Operation**

Selects `item`. If the operation succeeds, returns `true`; otherwise, returns `false`.

**deselect(archive item)**

**Declaration**

```dxl
bool deselect(ArchiveItem item)
```

**Operation**

Deselects `item`. If the operation succeeds, returns `true`; otherwise, returns `false`.

**rename(archive item)**

**Declaration**

```dxl
bool rename(ArchiveItem item, string newName)
```

**Operation**

Renames `item` to `newName`. If the operation succeeds, returns `true`; otherwise, returns `false`. 
get(archive data)

**Declaration**

```dxl
string get(string fileName,
            ArchiveData &archive,[bool serverArchive])
```

**Operation**

Retrieves the archive data structure from the given file. If the operation succeeds, returns a null string; otherwise, returns an error message.

The `serverArchive` flag indicates that the archive file is on the server.

If `serverArchive` flag is set to `true` and the user does not have permission to read a server archive, an error message will be returned.

canCreateServerArchive

**Declaration**

```dxl
bool canCreateServerArchive()
```

**Operation**

Returns `true` if the current user has permission to create an archive at the server and the server has been set up with the archive directory defined.

canRestoreServerArchive

**Declaration**

```dxl
bool canRestoreServerArchive()
```

**Operation**

Returns `true` if the current user has permission to restore an archive from the server and the server has been set up with the archive directory defined.

canReadServerArchiveFile

**Declaration**

```dxl
string canReadServerArchiveFile(string s)
```

**Operation**

Used to test for an archive existing and being available on the server before attempting a restore operation. The filename supplied is relative to the archive directory on the server.
Returns NULL if the file exists and can be read or a non-null error message if the specified file cannot be accessed on the server.

---

canWriteServerArchiveFile

declaration

```
string canWriteServerArchiveFile(string s)
```

operation

Used to test for an archive being available to write to on the server before attempting an archive operation. The filename supplied is relative to the archive directory on the server.

Returns NULL if the file can be written or a non-null error message if the specified file cannot be written to on the server. Overwriting server archives is not permitted so if the file already exists, this will return an error message.

---

canUseServerArchive

Declaration

```
string canUseServerArchive()
```

Operation

Used to return a string indicating if server archiving by the current user is allowed.

Returns a string indicating if server archiving is permitted and, if it is not permitted, why it is not.

The reasons server archiving may not be permitted are:

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server archiving not allowed - directory not defined</td>
<td>DOORS_ARCHIVE_LOCATION has not been defined</td>
</tr>
<tr>
<td>Server archive directory does not exist</td>
<td>The directory defined in DOORS_ARCHIVE_LOCATION does not exist or is not a sub-directory of SERVERDATA.</td>
</tr>
<tr>
<td>No permission to restore a server archive</td>
<td>The archive directory is valid but the user does not have permission to archive on the server.</td>
</tr>
</tbody>
</table>

If archiving is permitted, the following message is returned:

User has permission and directory is defined
Checksum validation

The checksum validation functions enable you to create a validation record for a module before you export or archive the module. The checksum function generates XML files in a compressed file with a .zip extension. The output identifies a module name, description, and location. The output also lists selected views for the module and the object attributes that are used in those views. Links, OLE objects, and images are not referenced in the generated output. When the module is later imported or restored to the project, you can then compare the checksum validation record with the module to identify changes to the text or other attribute values in the selected views.

createChecksumFile

Declaration

bool createChecksumFile(DB box, Module mod, Skip viewList, string fileName, string& returnStr)

Operation

Creates a compressed file with a .zip extension that contains XML files that describe a module, selected views within the module, and object attributes that are associated with objects in those views.

The createChecksumFile declaration includes the following parameters:

- The DB box parameter is optional. This parameter is used to display a progress bar while the checksum file is generated. If this parameter is not included in the declaration, then the progress bar is not shown.
- The Module mod parameter identifies the module from which the checksum file is generated.
- The Skip viewList parameter lists the views that are selected in the Create Checksum dialog box. At least one view is required to generate a checksum file.
- The string fileName parameter identifies the file path that is chosen by user for storing the checksum compressed file. If this field is empty or an invalid path, then a checksum file is not generated.
- The string& returnStr parameter is optional. This parameter contains errors that were encountered by the create operation, if any. If the operation completes successfully, the return value is an empty string.

The createChecksumFile function returns a boolean value to indicate if the create operation completed successfully. The value is True if the operation is successful. If the string& returnStr parameter is included, errors are displayed.

Example 1 - without progress bar

string modName = "myModuleName" // name of the module of your choice
Module modRef = read(modName, false)

int skipNo = 0
Skip viewListFromModule = create

// Enter the Standard View First
put(viewListFromModule, skipNo++, "Standard view")

//Now other views from module
string theViewName
for theViewName in views(modRef) do {
    // clear the noError
    put(viewListFromModule, skipNo++, theViewName)
}

string locationOfOutput = "C:\\temp\\checksumfile.zip" // or C:\temp

string createErrString
bool isSuccess = createChecksumFile(modRef, viewListFromModule, locationOfOutput, createErrString)

Example 2 - with progress bar
// Use definition of module, skip, and string from Example 1
// Include optional parameter to display progress bar:
DB box = create("Create Checksum ", styleCentred|styleAutoparent|styleThemed)
realize box
bool isSuccess = createChecksumFile(box, modRef, viewListFromModule, locationOfOutput, createErrString)

loadChecksumFile

Declaration

bool loadChecksumFile(Module mod, Skip& outViewList, string fileName, string& returnStr)

Operation

The loadChecksumFile function is preparation for the compareChecksumFile function. The loadChecksumFile function validates the module information and provides a list of the views that are available for the checksum comparison.

The loadChecksumFile declaration includes the following parameters:

- The Module mod parameter identifies the module that is associated with the checksum file.

- Skip& outViewList is the resulting out parameter. This list contains the views that are present in the checksum package. The user can choose one or more views from this list to run the checksum comparison on.
• The string fileName parameter contains the full path and file name of the stored checksum package, which is a compressed file with a .zip extension. The value must be a valid path and file.

• The string& returnStr parameter is optional. This parameter contains errors that were encountered by the create operation, if any. If the operation completes successfully, the return value is an empty string.

The loadChecksumFile function returns a boolean value to indicate if the load file operation completed successfully. The value is True if the operation was successful. If any error is found, then the return value is False. If the string& returnStr parameter is included, errors are displayed.

Example

```dxl
string modName = "myModuleName"  // name of the module of your choice
Module modRef = read(modName, false)
Skip outViewList = create
string checksumFile = "C:\temp\checksumfile.zip" //The valid link to existing checksum zip file.
string loadErrString
bool isSuccess = loadChecksumFile(modRef, outViewList, checksumFile, loadErrString)
```

**compareChecksumFile**

**Declaration**

```dxl
bool compareChecksumFile(DB box, Module mod, Skip viewList, string fileName, string& returnStr)
```

**Operation**

Enables the user to compare the checksum validation record with the related module to identify changes to the text or other attribute values in the selected views.

The compareChecksumFile declaration includes the following parameters:

• The DB box parameter is optional. If this parameter is included, the progress bar and Checksum Results dialog box are shown while the comparison process is running. The Checksum Results dialog box only shows comparison results. If the comparison process is not successful or completed, the error reasons are returned in the returnStr parameter.

• The Module mod parameter identifies the module that is associated with the checksum file. This parameter is passed to verify that the module in the Compare Checksum dialog box is the same as the module that is read from the XML content of the checksum package.

• The Skip skipViewList parameter provides the names of all the selected views in the Compare Checksum dialog box view list.

• The String fileName parameter specifies the full path and file name for checksum package, which is a compressed file with a .zip extension. If this parameter is NULL then the checksumPackageFullPath that is passed with the loadChecksumFile perm is used while comparing the views.
• The string & returnStr parameter is optional. If this parameter is included, it returns the error strings that indicate why the comparison process did not run or was not completed successfully. If the parameter DB box is not included, then the Checksum Results dialog box is not displayed and the comparison results are returned in this returnStr parameter.

The compareChecksumFile perm returns a boolean value. This value is True if the comparison operation completed successfully, even if some comparisons failed. The value is False if the comparison operation was not performed.

Example 1 - without the progress bar

string modName = "myModuleName"  // name of the module of your choice
Module modRef = read(modName, false)
Skip viewList = create
string checksumFile = "C:\\temp\\checksumfile.zip"  // The valid link to existing checksum zip file.
string loadErrString
bool isLoadSuccess = loadChecksumFile(modRef, viewList, checksumFile, loadErrString)
if ( isLoadSuccess) {
    string compareResults
    bool isCompareSuccess = compareChecksumFile(modRef, viewList, checksumFile, compareResults)
    if(isCompareSuccess) {
        print "/n compareResults = /n" compareResults "/n"
    }
}

Example 2 - with the progress bar

// Picking up after load Success from previous example
// Include optional parameter to display status messages:
if ( isLoadSuccess) {
    if ( isLoadSuccess) {
        string compareResults
        DB box = create("Compare Checksum ",
        styleCentred|styleAutoparent|styleThemed)
        realize box
        bool isSuccess = compareChecksumFile(box, modRef, viewList, checksumFile, compareResults)
Locking

This section defines functions that are used by the manipulation of data locks. They are rarely needed by normal DXL programs.

Most use the data types LockList and Lock.

**Note:** To obtain a type Lock handle, you must use the for lock in lock list loop.

Lock properties

Properties are defined for use with the . (dot) operator and a lock handle, as shown in the following syntax:

```
lock.property
```

where:

- `lock` Is a variable of type Lock
- `property` Is one of the lock properties

The value of `property` can be one of the following:

<table>
<thead>
<tr>
<th>String property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotation</td>
<td>Annotation associated with the lock</td>
</tr>
<tr>
<td>host</td>
<td>Host name to which the lock is assigned</td>
</tr>
<tr>
<td>id</td>
<td>Lock id, which distinguishes shared locks on an item</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the locked resource</td>
</tr>
<tr>
<td></td>
<td>For items in the module hierarchy, this is the unqualified name of the item.</td>
</tr>
<tr>
<td></td>
<td>For locks on the user list, this is User List(Read) or User List(Write). Separate read and write locks are used, for example, while archiving and restoring the user list.</td>
</tr>
<tr>
<td>user</td>
<td>The user account to which the lock is assigned</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Boolean property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>childLocked</td>
<td>Whether the lock is associated with a lock on a descendant of the locked item</td>
</tr>
<tr>
<td>removed</td>
<td>Whether the lock has been removed</td>
</tr>
</tbody>
</table>
All locks in a lock list are initially in either `lockShare` or `lockWrite` mode. To change them to `lockRemoved`, use the `remove(lock)` function.

**Example**

```dxl
Lock lockItem
string username
int connId
LockList lcklist = getLocksInDatabase(true)
for lockItem in lcklist do {
    username = lockItem.user
    connId = lockItem.connectionId
    print "User: " username ", Connection ID: " connId "\n"
}
```

### getLocksInDatabase

**Declaration**

```dxl
LockList getLocksInDatabase([bool allUsers])
```

**Operation**

Returns a lock list of type `LockList`, which lists lock information on locks held anywhere in the database except for locks on items that the user currently has open. If `allUsers` is true, the list contains locks held by all users. If `allUsers` is false, or missing, the list contains only locks held by the current user.
You must delete the LockList that is returned by this function before you run another function that returns a LockList.

Example
LockList llist
llist = getLocksInDatabase(true)

getLocksInFolder

Declaration
LockList
getLocksInFolder({Folder|Project} reference,
    bool recurse
    [,bool allUsers])

Operation
Returns a lock list of type LockList, which lists lock information on locks held anywhere in the folder or project reference. If recurse is true, the list contains all locks on descendants of the folder. If allUsers is true, the list contains locks held by all users. If allUsers is false, or missing, the list contains only locks held by the current user.

You must delete the LockList that is returned by this function before you run another function that returns a LockList.

Example
LockList llist
llist = getLocksInFolder(current project,true,true)

getLocksInModule

Declaration
LockList getLocksInModule(ModName_ modRef
    [,bool allUsers])

Operation
Returns a lock list of type LockList, which lists lock information on locks held anywhere in module modRef. If allUsers is true, the list contains locks held by all users. If allUsers is false, or missing, the list contains only locks held by the current user.

You must delete the LockList that is returned by this function before you run another function that returns a LockList.
isLocked

Declaration

bool isLocked(ModName_ modRef)

Operation

Returns true if the specified module is locked by a user; otherwise returns false.

Note that this function returns true even if a specified module is locked by the current user.

Example

print isLocked(module "New Module")

isLockedClosed

Declaration

bool isLockedClosed(ModName_ modRef)

Operation

Returns true if the current user has an exclusive lock on module \( m \), and the module is not currently open. Otherwise, returns false.

isLockedByUser

Declaration

bool isLockedByUser(Object o)

Operation

Returns true if the specified object is locked by the current user when in edit shareable mode. Otherwise, returns false.

This function is not equivalent to checking whether the current user can modify the given object.

lock(module)

Declaration

string lock(ModName_ modRef

[,string annotation])
Operation
Places an exclusive lock on module modRef, without opening it. Also places share locks on all of its ancestor folders (up to the nearest project). The optional second argument associates an annotation with the lock, which can be retrieved through the annotation property (see “Lock properties,” on page 931). If annotation is a null string or only white space characters, no annotation is stored with the lock.

If the user does not have modify, create, delete, or control access to modRef, the call fails.

If the operation succeeds, returns null; otherwise, returns an error message.

Example

```dxl
string errormess
errormess = lock(module "My module")
if (null errormess)
    print "My module locked.\n"
else
    print errormess "\n"
```

lock(object)

Declaration

```dxl
string lock(Object o [, bool& unavailable])
```

Operation

Locks object o. If supplied, the unavailable parameter is set to true if the section cannot be locked due to a lock not being available. This is usually because another client has locked the section. If the operation succeeds, returns null; otherwise, returns an error message.

This function only makes sense when o is in a module that has been opened shareable.

Example

```dxl
if (isShare current) {
    string mess = lock current Object
    if (!null mess)
        print "lock failed: " mess "\n"
}
```

unlock(module)

Declaration

```dxl
string unlock(ModName_ modRef)
```

Operation

Removes an exclusive lock placed on module m by the same user. Fails if the module is open or this user has no exclusive lock on it. Removes the associated share locks on ancestor folders.
If the operation succeeds, returns `null`; otherwise, returns an error message.

**Example**

```dxl
string errormess
errormess = unlock(module "My module")
if (null errormess)
    print "My module unlocked.\n"
else
    print errormess "\n"
```

---

### delete(lock list)

**Declaration**

```dxl
string delete(LockList list)
```

**Operation**

Frees up memory used by the variable `list`. If `list` is null, this function has no effect.

**Example**

```dxl
LockList myList = getLocksInDatabase
delete myList
```

---

### remove(lock)

**Declaration**

```dxl
string remove(Lock lock)
```

**Operation**

Attempts to remove `lock` from the database. Any associated locks in the lock list are also removed. Associated locks are locks on descendants of a folder, and associated locks on ancestor folders that are not associated with locks on other descendants.

If the operation succeeds, returns a null string; otherwise, returns an error message.

---

### shareLock

**Declaration**

```dxl
string shareLock({Folder|Project} reference,
    string &lockID
    [,string annotation])
```
Operation

Places a share lock on the folder or project reference, until it is removed by the remove (lock) function. It does not lock ancestor folders. It passes back the lock ID in the second argument. The optional third argument associates an annotation with the lock, which can be retrieved through the annotation property (see “Lock properties,” on page 931).

If the operation succeeds, returns a null string; otherwise, returns an error message.

for lock in lock list

Syntax

for lock in list do {
    ...
}

where:

lock Is a variable of type Lock
list Is a variable of type LockList

Operation

Assigns the variable lock to be each successive lock in list.

Example

Lock lockItem
string username
LockList lcklist = getLocksInDatabase(true)
for lockItem in lcklist do {
    username = lockItem.user
    print username "\n"
}

Unlock object functions

Declaration

bool unlockDiscard{All|Section}(Object o)
bool unlockSave{All|Section}(Object o)

Operation

These functions unlock sections. The functions unlockDiscardAll and unlockSaveAll unlock all sections in the module containing o. The functions unlockDiscardSection and unlockSaveSection unlock the section containing o.

The functions either discard changes or save changes before unlocking according to the function name.
If the operation is successful, returns true; otherwise, returns false.

requestLock

Declaration

string requestLock(Module m, Object o, bool exclusive, string msg, bool alert)
string requestLock(Module m, bool exclusive, string msg, bool alert)
string requestLock(Object o, string msg, bool alert)

Operation

The first form places a lock request on the specified module/object in the specified lock mode. If exclusive is set to true, an exclusive lock will be requested, otherwise a share lock will be requested. msg is the message (if any) to be sent.
The second form requests a lock on the module itself.
The third form requests a lock on the section containing the specified object.
All return errors on failure.

HTML functions

This section defines functions that create HTML to represent a Rational DOORS object attribute, and set an attribute value based on HTML.

htmlText

Declaration

string htmlText(Buffer &htmlOutput, Column c, Object o, bool showURL, bool newWin, string preLink, string postLink)
string htmlText(Buffer &htmlOutput, attrRef, bool showURL, bool newWin, string preLink, string postLink)

where attrRef is in one of the following formats:
(Object o).(string attrName)
Operation

The first form fills the buffer htmlOutput with a fragment of HTML representing the object o in column c. The second form does the same for the given object attribute.

The argument showURL controls whether URLs in the text are shown as hyperlinks. The argument newWin controls whether the hyperlinks should open a new browser window.

If showURL is true, the strings preLink and postLink contain text that appears before the hyperlink and after the hyperlink respectively.

If the call succeeds, returns a null string; otherwise, returns an error message.

Example

Buffer b = create
Object o = current Object
htmlText(b, o."Object Text", true, false, ",", ")
print b"\n"

setAttrFromHTML

Declaration

string setAttrFromHTML(Buffer &html,
                       attrRef,
                       where attrRef is in one of the following formats:

(Object o).(string attrName)
(Module m).(string attrName)
(Link l).(string attrName)

Operation

Sets the value of the specified attribute based on the HTML in the buffer.

If the call succeeds, returns a null string; otherwise, returns an error message.

Example

Buffer b = create
Object o = current Object
b = "hello <b>world</b>"
setAttrFromHTML(b, o."Object Text")
HTTP Server

This section defines functions for making HTTP requests to a URL.

Sample files are included in the Rational DOORS client installation. The sample file `hierarchy.dxl` shows how to use the OSLC DXL service to obtain information about module hierarchy. The sample file `sample_get.dxl` demonstrates other HTTP Server functions. The default location of the sample files is `C:\Program Files\IBM\Rational\DOORS\9.version\lib\dxl\example\oslc`. The path and version number might vary in your installation.

HttpRequest

Declaration

```dxl
HttpResponse HttpRequest(HttpVerb verb, string url, HttpBody body, HttpHeaders header)
```

Operation

This function makes the HTTP request to the provided URL. If the URL is to a server that is configured for Open Services for Lifecycle Collaboration (OSLC) the request is authenticated with OAuth, and automatically includes the configured OSLC version in the request header, as OSLC-CORE-VERSION. To determine if the server is configured for OSLC, see the Database Properties > Collaboration tab in the Rational DOORS client. The body and header parameters are optional.

Example

This example demonstrates access to a public URL with no OAuth authentication. The use of HttpHeaders is demonstrated, but is optional and can be null. After the request is sent, a response is returned. The example reads the HTML code from the response body and displays it. Optional cleaning of the memory is also demonstrated.

```dxl
HttpHeader h = create
string k = "Accept"
string v = "*/*
"
if (header != null)
{
  add(h, k, v)
}
HttpResponse resp = HttpRequest(HttpGet, "https://www.google.com", null, h)
if (resp != null && resp.isOk)
{
  HttpBody b = resp.body
  Buffer buf = b.value
  ack "Request succesfull\n"
  ack stringOf(buf)
  delete buf
}
```
else
{
    ack "Request failed"
}
delete h
delete resp

HttpRequest(timeout)

Declaration

HttpResponse HttpRequest(HttpVerb verb, string url, HttpBody body, HttpHeader header, int timeout)

Operation

This function is the same as httpResponse HttpRequest(HttpVerb, string, HttpBody, HttpHeader), but it includes an additional timeout argument. The request is performed as usual but with the specified timeout instead of the default one. The timeout argument is in seconds.

HttpRequest(error message)

Declaration

string HttpRequest(HttpVerb, string, HttpBody, HttpHeader, HttpResponse&)  

Operation

This function is the same as httpResponse HttpRequest(HttpVerb, string, HttpBody, HttpHeader), but it returns an error message in case of failure.

HttpRequest(timeout and error message)

Declaration

string HttpRequest(HttpVerb, string, HttpBody, HttpHeader, int, HttpResponse&)  

Operation

This function is the same as HttpResponse HttpRequest(HttpVerb verb, string url, HttpBody body, HttpHeader header, int timeout), but it returns an error message in case of failure.
HttpVerb

The HttpVerb is an enumeration that represents the type of HTTP request:

- HttpMethodDelete
- HttpMethodGet
- HttpMethodHead
- HttpMethodPut
- HttpMethodPost
HttpResponse

The HttpResponse object represents the response to an HTTP request.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpHeaders()</td>
<td>An HttpHeaders object that stores a map of key-value string pairs.</td>
</tr>
<tr>
<td>HttpBody()</td>
<td>Any data returned by the HTTP request in its body as an HttpBody object.</td>
</tr>
<tr>
<td>int code()</td>
<td>The HTTP response code.</td>
</tr>
<tr>
<td>bool isOk()</td>
<td>True for a successful HTTP request; for example, if the return code is 200 or 201.</td>
</tr>
</tbody>
</table>

Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void delete(HttpResponse&amp;)</td>
<td>Call to delete the HttpResponse object that is returned by HttpRequest. This operation is not mandatory.</td>
</tr>
</tbody>
</table>

HttpHeader

The HttpHeader object represents a collection of header key-value of a HTTP request or response.

Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string get(HttpHeader header, string key)</td>
<td>Returns a header pair value by its key.</td>
</tr>
<tr>
<td>void add(HttpHeader&amp; header, string key, string value)</td>
<td>Add a header key-value pair to header.</td>
</tr>
<tr>
<td>void add(HttpHeader&amp; header, HttpHeaderEntry entry)</td>
<td>Add a header key-value pair to header from HttpHeadersEntry.</td>
</tr>
<tr>
<td>HttpHeaders create()</td>
<td>Call to create a HttpHeaders object.</td>
</tr>
<tr>
<td>void delete(HttpHeader&amp;)</td>
<td>Call to delete the HttpHeaders object returned by create, this isn't mandatory.</td>
</tr>
</tbody>
</table>
Iterators

<table>
<thead>
<tr>
<th>Iterator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>for HttpHeadersEntry in HttpHeaders do</td>
<td>Returns all HttpHeadersEntry in the header. The order is undetermined.</td>
</tr>
</tbody>
</table>

HttpHeaderEntry

HttpHeaderEntry represents a header key-value pair.

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>string value()</td>
<td>The HttpHeadersEntry value.</td>
</tr>
<tr>
<td>string key()</td>
<td>The HttpHeadersEntry key.</td>
</tr>
</tbody>
</table>

Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void delete(HttpHeadersEntry&amp;)</td>
<td>Call to delete the HttpHeadersEntry object that is returned by create. This operation is not mandatory.</td>
</tr>
<tr>
<td>HttpHeadersEntry create()</td>
<td>Call to create a HttpHeadersEntry object.</td>
</tr>
<tr>
<td>void setValue(HttpHeadersEntry&amp; entry, string value)</td>
<td>Set the HttpHeadersEntry value.</td>
</tr>
<tr>
<td>void setKey(HttpHeadersEntry&amp; entry, string value)</td>
<td>Set the HttpHeadersEntry key.</td>
</tr>
</tbody>
</table>

HttpBody

The HttpHeaders object represents the body (data part) of an HTTP request or response.
Asynchronous HTTP requests

These functions are used to make HTTP requests asynchronously so that the main Rational DOORS process is not blocked. With these functions, the DXL user interface is still be responsive while an HTTP request is in progress.

Future HttpRequest

Declaration

Future HttpRequest(HttpVerb verb, string url, HttpBody b, HttpHeaders h)

Where Future is an object that contains a value that will be delivered in the future.

Operation

This function starts an HTTP request but instead of waiting for a response, it immediately returns a Future object. The Future object can be checked for readiness later. The HTTP response can be fetched from the Future object when it is ready.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool isReady(Future f)</td>
<td>Returns true if the Future is delivered.</td>
</tr>
<tr>
<td>int getFullfilledTime(Future f)</td>
<td>Returns the timestamp for the Future's delivery time.</td>
</tr>
</tbody>
</table>
Helper operations for using the asynchronous HTTP request perm in layout DXL columns:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
</table>
| void refresh(Future f, Object o, Column c, int elapsed) | Re-executes the layout DXL for a cell after the Future is fulfilled. After the Future is delivered, this operation:  
  • Invalidates the displayable object cache for the cell (Object o, Column c).  
  • After 'elapsed' seconds, refreshes everything in the current view set. This causes the layout DXL execution, because the cell does not have a displayable object cache. |
| Future getData(Object o, string key)            | Retrieves a Future.                                                                                   |
| void setData(Object o, string key, Future f)    | Puts a Future inside object's "client data". This keeps the Future for successive layout DXL executions. |
| void removeData(Object o, string key)           | Removes a Future from object's "client data".                                                        |

Dialog box example:

```java
// Async http request example with dxl timer
DB box
DBE htmlViewElem
DBE timerElem

string url = "http://en.m.wikipedia.org/wiki/DOORS"

// Make an async request put the future in object client data
Future f = HttpRequest(HttpGet, url, null, null)

// Will be called in every second,
// when the response is delivered it will stop the timer.
void onTimer(DBE)
```
{  
  if (isReady(f))
  {
    HttpResponse r = getValueHttpResponse(f)
    set(htmlViewElem, r.body.value)
    stopTimer(timerElem)
  }
  else
  {
    Buffer b = create
    b = "<h3>LOADING URL: " url "</h3>"
    set(htmlViewElem, b)
    delete b
  }
}

// Prepare a dialog with a timer
box = create "Async HttpRequest Example!"
htmlViewElem = htmlEdit(box, "", 600, 400)
timerElem = timer(box, 1, onTimer, "check response timer!")
show box

Layout DXL example:

  // This is a layout DXL example. For each objects in the module, it fetches the 'url' and displays its status code.
  
  // Usage: Add a column via "Edit -> Columns -> New... -> Layout DXL -> Browse -> New -> (Paste this)"
  Column c = currentColumn
  string url = "http://www.ibm.com/robots.txt"

  void startRequest(string url)
  {
    Future f = HttpRequest(HttpGet, url, null, null)
    setData(obj, "KEY_PREF_" url, f)
refresh(f, obj, c, 5 /*seconds timeout*/)
display("Started request:\n\turl "\n")
}

Future f =getData(obj, "KEY_PREF_" url)
if (null f) // No request made yet
{
    startRequest(url)
}
else if (isReady(f))
{
    if (intOf today - getFullfilledTime(f) > 30) // Ready but expired (> 30 seconds)
    {
        removeData(obj, "KEY_PREF_" url)
        startRequest(url)
    }
    else // Ready and fresh response!
    {
        HttpResponse r = getValueHttpResponse(f)
        display("Response received with status code: " r.code "\n")
    }
}
else // Waiting for a response!
{
    display("Started request:\n\turl "\n")
}
OSLC DXL Services

OSLC DXL Services are DXL scripts that can be run by making an Open Services for Lifecycle Collaboration (OSLC) request to an instance of Rational DOORS Web Access. A service must be added to the DOORS database before it can be run. DXL functions are available to add, remove, and return information about DXL services. For more information and examples, see the help topic: OSLC DXL services for Rational DOORS.

OSLCDXLServicelService properties

Properties are defined for use with the . (dot) operator and a OSLCDXLServicelService handle, as shown in the following syntax:

variable.property

where:

variable Is a variable of type OSLCDXLServicelService

property Is one of the OSLCDXLServicelService properties

The following table lists the OSLCDXLServicelService properties and the information that they extract or specify.

For further details on specifying information, see the setDxIServiceResult function.
Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Extracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the service, as displayed by OSLC service discovery.</td>
</tr>
<tr>
<td>description</td>
<td>The explanation of what the service does. This can be returned by using the GET function on the service.</td>
</tr>
<tr>
<td>dxlString</td>
<td>Either a fragment of DXL that can be run or a collection of functions, one of which can be called by running &quot;functionName&quot;.</td>
</tr>
<tr>
<td>functionName</td>
<td>The name of a function (in dxlString) to run. If this is NULL, then dxlString will be treated as runnable DXL.</td>
</tr>
</tbody>
</table>

Iterators

<table>
<thead>
<tr>
<th>Iterator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>for OSLCDXLService in database do</td>
<td>Loop that iterates over available DXL services.</td>
</tr>
</tbody>
</table>

setDxlServiceResult

**Declaration**

```c
void setDxlServiceResult(string result)
```

**Operation**

Sets the result string that is returned as a result of the service that is being run.

**Example**

For more information and an example, see the help topic: OSLC DXL services for Rational DOORS.

addOrUpdateOSLCDXLService

**Declaration**

```c
string addOrUpdateOSLCDXLService(string name, string description, string dxlString, string functionName)
```

**Operation**

Adds a new service to the list of configured DXL services or updates an existing one.

**Example**

For more information and an example, see the help topic: OSLC DXL services for Rational DOORS.
removeOSLCDXLService(string key)

Declaration
string removeOSLCDXLService(string key)

Operation
Removes a service from the configuration. Finds the service by its key (that is, name).

removeOSLCDXLService(OSLCDXLService service)

Declaration
string removeOSLCDXLService(OSLCDXLService service)

Operation
Removes a service object from the configuration.

Broadcast Messaging

sendBroadcastMessage

Declaration
string sendBroadcastMessage(string msg)

Operation
Sends a message to the database server for broadcasting to all connected clients. Returns an error string if broadcasting fails, otherwise returns null. The executing user must have the Manage Database privilege.

Example
if (null sendBroadcastMessage("Please save your work and logout immediately.")){
  ack "Message sent"
}
Converting a symbol character to Unicode

symbolToUnicode

Declaration

char symbolToUnicode(char symbolChar, bool convertAllSymbols)

Operation

Converts a symbol character to its Unicode equivalent. If \textit{convertAllSymbols} is false, only symbols with the Times New Roman font equivalents are converted. This function returns a unicode character, or \texttt{charIn} if there is no equivalent character.
Timer

timer

Declaration
DBE timer(DB parent, [int/real] timer_duration_in_seconds, void(DBE) callbackFn, string timer_name)

Operation
Creates a timer element that executes the callback function every 'n' seconds.

Example
The following example creates a timer with a two-second interval and callback cb that prints a message.

```c
void cb(DBE x){
    print "timer event,"
}

DBE dbeTime = timer(dbParent, 2, cb, "")
```

stopTimer

Declaration:
bool stopTimer(DBE timer)

Operation
Stops the execution of a specific timer. Returns true if the timer was running and is now stopped.

Example
This example stops timer.

```c
stopTimer(dbeTimer)
```

startTimer

Declaration
bool startTimer(DBE timer)

Operation
Restarts the execution of a specific timer. Returns true if the timer was stopped and is now restarted.
Example
This example restarts the timer.
startTimer(dbeTimer)

isTimer

Declaration
bool isTimer(int id)

Operation
Returns TRUE if id is a valid action index and it is a timer.

getTimerName

Declaration
string getTimerName(int id)

Operation
Returns a string containing the name of the timer (or NULL if not a timer).

getTimerInterval

Declaration
int getTimerInterval(int id)

Operation
Returns the number of seconds between each execution of the timer.

getTimerID

Declaration
int getTimerID(DBE t)

Operation
Returns the timer t ID or -1 if t is not a timer.
Example

The following example prints "test".

```dxl
void cb(DB x) {}
DB db = create("test")
DBE t = timer(db, 1, cb, "test")
realize db
int id = getTimerID(t)
print getTimerName id
```

---

**getTimerRunning**

**Declaration**

```dxl
bool getTimerRunning(int id)
```

**Operation**

Returns TRUE if `id` is a timer and it is running.

**Example**

The following example prints all running timers.

```dxl
int i
for i in 0 : 999 do
{
  string s = getTimerName(i)
  if (!null s)
  {
    int    d = getTimerInterval(i)
    bool   r = getTimerRunning (i)
    print "Timer " i " is " (r ? "" : "not ") "running with interval " d "\n"
  }
}
```
# Chapter 37

## Character codes and their meanings

The following table lists the characters for ASCII codes 0-127. For ASCII codes 128 and higher, Rational DOORS uses Latin-1 encoding. The character sets for Latin-1 differ between platforms.

### Table 1: ASCII codes 0-127

<table>
<thead>
<tr>
<th>Decimal Value</th>
<th>Hex Value</th>
<th>Control Character</th>
<th>ASCII Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>Ctrl-@</td>
<td>NUL</td>
<td>null</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>Ctrl-A</td>
<td>SOH</td>
<td>start of heading</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>Ctrl-B</td>
<td>STX</td>
<td>start of text</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>Ctrl-C</td>
<td>ETX</td>
<td>end of text</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>Ctrl-D</td>
<td>EOT</td>
<td>end of transmission</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>Ctrl-E</td>
<td>ENQ</td>
<td>enquiry</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>Ctrl-F</td>
<td>ACK</td>
<td>acknowledge</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>Ctrl-G</td>
<td>BEL</td>
<td>bell</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>Ctrl-H</td>
<td>BS</td>
<td>backspace</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>Ctrl-I</td>
<td>HT</td>
<td>horizontal tab</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>Ctrl-J</td>
<td>LF</td>
<td>new line</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>Ctrl-K</td>
<td>VT</td>
<td>vertical tab</td>
</tr>
<tr>
<td>12</td>
<td>OC</td>
<td>Ctrl-L</td>
<td>FF</td>
<td>form feed</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>Ctrl-M</td>
<td>CR</td>
<td>carriage return</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>Ctrl-N</td>
<td>SO</td>
<td>shift out</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>Ctrl-O</td>
<td>SI</td>
<td>shift in</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>Ctrl-P</td>
<td>DLE</td>
<td>data link escape</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>Ctrl-Q</td>
<td>DC1</td>
<td>device control 1</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>Ctrl-R</td>
<td>DC2</td>
<td>device control 2</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>Ctrl-S</td>
<td>DC3</td>
<td>device control 3</td>
</tr>
<tr>
<td>Decimal Value</td>
<td>Hex Value</td>
<td>Control Character</td>
<td>ASCII Symbol</td>
<td>Meaning</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>-------------------</td>
<td>--------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>Ctrl-T</td>
<td>DC4</td>
<td>device control 4</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>Ctrl-U</td>
<td>NAK</td>
<td>negative acknowledge</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>Ctrl-V</td>
<td>SYN</td>
<td>synchronous idle</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>Ctrl-W</td>
<td>ETB</td>
<td>end of transmission block</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>Ctrl-X</td>
<td>CAN</td>
<td>cancel</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>Ctrl-Y</td>
<td>EM</td>
<td>end of medium</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>Ctrl-Z</td>
<td>SUB</td>
<td>substitute</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>Ctrl-[</td>
<td>ESC</td>
<td>escape</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>Ctrl-\</td>
<td>FS</td>
<td>file separator</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>Ctrl-]</td>
<td>GS</td>
<td>group separator</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>Ctrl-^</td>
<td>RS</td>
<td>record separator</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>Ctrl-_</td>
<td>US</td>
<td>unit separator</td>
</tr>
<tr>
<td>32</td>
<td>20</td>
<td>SP</td>
<td></td>
<td>digit select</td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>!</td>
<td></td>
<td>exclamation point</td>
</tr>
<tr>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td></td>
<td>double quotation mark</td>
</tr>
<tr>
<td>35</td>
<td>23</td>
<td>#</td>
<td></td>
<td>pound sign, number sign</td>
</tr>
<tr>
<td>36</td>
<td>24</td>
<td>$</td>
<td></td>
<td>dollar sign</td>
</tr>
<tr>
<td>37</td>
<td>25</td>
<td>%</td>
<td></td>
<td>percent sign</td>
</tr>
<tr>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td></td>
<td>ampersand</td>
</tr>
<tr>
<td>39</td>
<td>27</td>
<td>'</td>
<td></td>
<td>apostrophe</td>
</tr>
<tr>
<td>40</td>
<td>28</td>
<td>(</td>
<td></td>
<td>left parenthesis</td>
</tr>
<tr>
<td>41</td>
<td>29</td>
<td>)</td>
<td></td>
<td>right parenthesis</td>
</tr>
<tr>
<td>42</td>
<td>2A</td>
<td>*</td>
<td></td>
<td>asterisk</td>
</tr>
<tr>
<td>43</td>
<td>2B</td>
<td>+</td>
<td></td>
<td>addition sign</td>
</tr>
<tr>
<td>44</td>
<td>2C</td>
<td>,</td>
<td></td>
<td>comma</td>
</tr>
</tbody>
</table>
Table 1: ASCII codes 0-127

<table>
<thead>
<tr>
<th>Decimal Value</th>
<th>Hex Value</th>
<th>Control Character</th>
<th>ASCII Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>2D</td>
<td></td>
<td>-</td>
<td>subtraction sign</td>
</tr>
<tr>
<td>46</td>
<td>2E</td>
<td></td>
<td>.</td>
<td>period</td>
</tr>
<tr>
<td>47</td>
<td>2F</td>
<td></td>
<td>/</td>
<td>right slash</td>
</tr>
<tr>
<td>48</td>
<td>30</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>31</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>32</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>33</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>34</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>35</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>36</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>37</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>38</td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>39</td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>3A</td>
<td></td>
<td>:</td>
<td>colon</td>
</tr>
<tr>
<td>59</td>
<td>3B</td>
<td></td>
<td>;</td>
<td>semicolon</td>
</tr>
<tr>
<td>60</td>
<td>3C</td>
<td></td>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>61</td>
<td>3D</td>
<td></td>
<td>=</td>
<td>equal</td>
</tr>
<tr>
<td>62</td>
<td>3E</td>
<td></td>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>63</td>
<td>3F</td>
<td></td>
<td>?</td>
<td>question mark</td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td></td>
<td>@</td>
<td>at sign</td>
</tr>
<tr>
<td>65</td>
<td>41</td>
<td></td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>42</td>
<td></td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>43</td>
<td></td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>44</td>
<td></td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>45</td>
<td></td>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>
Table 1: ASCII codes 0-127

<table>
<thead>
<tr>
<th>Decimal Value</th>
<th>Hex Value</th>
<th>Control Character</th>
<th>ASCII Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>46</td>
<td></td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>47</td>
<td></td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>48</td>
<td></td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>49</td>
<td></td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>4A</td>
<td></td>
<td>J</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>4B</td>
<td></td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>4C</td>
<td></td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>4D</td>
<td></td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>4E</td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>4F</td>
<td></td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>50</td>
<td></td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>51</td>
<td></td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>52</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>53</td>
<td></td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>54</td>
<td></td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>55</td>
<td></td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>56</td>
<td></td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>57</td>
<td></td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>58</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>59</td>
<td></td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>5A</td>
<td></td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>5B</td>
<td></td>
<td>[</td>
<td>left bracket</td>
</tr>
<tr>
<td>92</td>
<td>5C</td>
<td></td>
<td>\</td>
<td>left slash, backslash</td>
</tr>
<tr>
<td>93</td>
<td>5D</td>
<td></td>
<td>]</td>
<td>right bracket</td>
</tr>
<tr>
<td>94</td>
<td>5E</td>
<td></td>
<td>^</td>
<td>hat, circumflex, caret</td>
</tr>
</tbody>
</table>
Table 1: ASCII codes 0-127

<table>
<thead>
<tr>
<th>Decimal Value</th>
<th>Hex Value</th>
<th>Control Character</th>
<th>ASCII Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>95</td>
<td>5F</td>
<td></td>
<td>_</td>
<td>underscore</td>
</tr>
<tr>
<td>96</td>
<td>60</td>
<td></td>
<td>`</td>
<td>grave accent</td>
</tr>
<tr>
<td>97</td>
<td>61</td>
<td></td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>62</td>
<td></td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>63</td>
<td></td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>64</td>
<td></td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>65</td>
<td></td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>66</td>
<td></td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>67</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>68</td>
<td></td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>69</td>
<td></td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>6A</td>
<td></td>
<td>j</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>6B</td>
<td></td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>6C</td>
<td></td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>6D</td>
<td></td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>6E</td>
<td></td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>6F</td>
<td></td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>70</td>
<td></td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>71</td>
<td></td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>72</td>
<td></td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>73</td>
<td></td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>74</td>
<td></td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>75</td>
<td></td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>76</td>
<td></td>
<td>v</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>77</td>
<td></td>
<td>w</td>
<td></td>
</tr>
</tbody>
</table>
Table 1: ASCII codes 0-127

<table>
<thead>
<tr>
<th>Decimal Value</th>
<th>Hex Value</th>
<th>Control Character</th>
<th>ASCII Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>78</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>79</td>
<td></td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>7A</td>
<td></td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>7B</td>
<td></td>
<td>{</td>
<td>left brace</td>
</tr>
<tr>
<td>124</td>
<td>7C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>7D</td>
<td></td>
<td>}</td>
<td>right brace</td>
</tr>
<tr>
<td>126</td>
<td>7E</td>
<td></td>
<td>~</td>
<td>similar, tilde</td>
</tr>
<tr>
<td>127</td>
<td>7F</td>
<td></td>
<td>DEL</td>
<td>delete</td>
</tr>
</tbody>
</table>
Chapter 38

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send written license inquiries to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send written inquiries to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions. Therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, Massachusetts 01886
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at www.ibm.com/legal/copytrade.html.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

PostScript is either a registered trademark or trademark of Adobe Systems Incorporated in the United States, and/or other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names may be trademarks or service marks of others.
Index

Symbols

:: operator 17
::do(AccessRec&, SignatureInfo, void) 353
::do(AccessRec&, SignatureInfoSpecifier__, void) 353
::do(SignatureEntry&, SignatureInfo, void) 357

A

accept(IPC) 189
accept(object) 647
acceptPartition 716
acceptReport 716
Access status 470
accessed(date) 182
AccessRec get(SignatureInfo, string name, string& error) 354
acknowledge 486
activateURL function 26
active(element) 503
Actual colors 593
addAcceleratorKey 489
addAttribute 710
addAwayAttribute 710
addAwayLinkModule 709
addAwayLinkset 711
addAwayModule 709
addAwayView 711
addBaselines(BaselineSet) 326
addComment(Discussions) 901
addFilter 647
addGroup 247
addGroup (Discussions access controls) 907
addLinkModule 708
addLinkModuleDescriptor 404
addLinkset 711
addMember 247
addModule 708
addModule(BaselineSetDefinition) 315
addNotifyUser 224
addOrUpdateOSLCDXLService 950
addProjectEntry(ProblemItem) 894
addRecentlyOpenModule(ModuleVersion) 344
addRecentlyOpenModule(string) 345
addToolTip 578
addUser 246
addUser (Discussions access controls) 907
addView 711
all(module) 372
allAttributesReadable(SignatureEntry) 360
allowsAccess 713
alternative 864, 870
Alternative Addins Location 627
alternatives 864
alternativeWord 880
anagram 882
ancestors(show/hide) 654
ancestors(state) 654
annotation(BaselineSet) 324
append 207
Append operator (BaselineSet) 324
append(open file) 135
appendCell 822
appendColumn(table) 822
appendRow 823
appendSignatureEntry(SignatureInfo si, string label, string comment) 356
Application of regular expressions 159
apply 557
apply(arrows) 559
apply(partition definition) 715
applyFiltering 655
applyingFiltering 655
applyTextFormattingToParagraph 849
Archive
and restore 914
properties 914
archive(modules and projects) 916
archive(user list) 917
archiveFiles 917
Arithmetic operators (int) 119
Arithmetic operators (real) 122
Arrays 172
ascending 661
Assignment (buffer) 163
Assignment (date) 148
Assignment (enumerated option) 438
Assignment (from attribute) 430
Assignment (int) 120
Assignment (real) 123
Assignment (rich text) 840
Assignment (to attribute) 431
Asynchronous HTTP requests 945
Attachment placement 601
attrdef(trigger) 786
attrDXLName 466
Attribute definitions
  access controls 450
  example program 448
  introduction 439
  properties 41, 49, 439
Attribute types
  access controls 458
  manipulation 459
  properties 452
  values 50, 452
Attribute values
  access controls 436
  extraction 429
attribute(in column) 688
attribute(trigger) 786
attribute(value) 646
attributeDXL 550
attributeValue 201, 444
attrName 689
Auto-declare 8
autoIndent 302
Automation
  interface 771
Automation client 766

B
background 562
backgroundColor(get) 689
backgroundColor(set) 690
backSlasher 129
Backtrace 912
baseline 305
baseline(history session) 337
baseline(ModuleVersion) 311
baselineExists 305
baselineExists(ModuleVersion) 311
baselineIndex(discussions) 904
baselineInfo(current Module) 311
Baselines, example program 309
baselineSet(ModuleVersion) 327
baseWin 490
batchMode function 26
below(element) 596
beside 596
bitmap 571
block 490
bool pageSignaturePage 814

Boolean operators 116
box 566
Break statement 22
bringToFront 304
Browsing the DXL library 5
Buffer comparison 164
Buffers and regular expressions 170
busy 490
button 558
button(arrows) 559
Buttons 556

C
Callstack 912
canceled(IntegrityResultsData) 889
cancelled(IntegrityResultsData) 889
canControl(attribute type) 458
canControl(attribute) 436
canControl(item) 266
canControl(module) 289
canControl(object) 370
canControl(view) 676
canControlDef 450
canControlVal 451
canCreate(attribute type) 458
canCreate(attribute) 436
canCreate(item) 266
canCreate(module) 289
canCreate(object) 370
canCreate(view) 676
canCreateAttrs 451
canCreateAttrTypes 459
canCreateDef 450
canCreateServerArchive 925
canCreateVal 450
canDelete 382
canDelete(attribute type) 459
canDelete(attribute) 437
canDelete(external link) 414
canDelete(item) 267
canDelete(link) 395
canDelete(module) 290
canDelete(object) 370
canDelete(view) 677
canDeleteDef 451
canDeleteVal 451
canEveryoneModifyDiscussions 906
canLock(object) 371
canModify(attribute type) 459
<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>canModify(attribute)</td>
</tr>
<tr>
<td>canModify(item)</td>
</tr>
<tr>
<td>canModify(module)</td>
</tr>
<tr>
<td>canModify(object)</td>
</tr>
<tr>
<td>canModify(view)</td>
</tr>
<tr>
<td>canModifyDiscussions</td>
</tr>
<tr>
<td>canOpenFile</td>
</tr>
<tr>
<td>canRead(attribute type)</td>
</tr>
<tr>
<td>canRead(attribute)</td>
</tr>
<tr>
<td>canRead(item)</td>
</tr>
<tr>
<td>canRead(view)</td>
</tr>
<tr>
<td>canReadServerArchiveFile</td>
</tr>
<tr>
<td>canRestoreServerArchive</td>
</tr>
<tr>
<td>canUnlock(object)</td>
</tr>
<tr>
<td>canUseServerArchive</td>
</tr>
<tr>
<td>canvas</td>
</tr>
<tr>
<td>canWrite(attribute)</td>
</tr>
<tr>
<td>canWrite(module)</td>
</tr>
<tr>
<td>canWrite(view)</td>
</tr>
<tr>
<td>canWriteServerArchiveFile</td>
</tr>
<tr>
<td>Casts</td>
</tr>
<tr>
<td>cell</td>
</tr>
<tr>
<td>centered</td>
</tr>
<tr>
<td>change(view definition)</td>
</tr>
<tr>
<td>changed(date)</td>
</tr>
<tr>
<td>changePasswordDialog</td>
</tr>
<tr>
<td>changePicture</td>
</tr>
<tr>
<td>Character constants</td>
</tr>
<tr>
<td>Character schemes</td>
</tr>
<tr>
<td>color</td>
</tr>
<tr>
<td>Color schemes</td>
</tr>
<tr>
<td>Colors</td>
</tr>
<tr>
<td>column</td>
</tr>
<tr>
<td>Column alignment constants</td>
</tr>
<tr>
<td>Columns</td>
</tr>
<tr>
<td>combine</td>
</tr>
<tr>
<td>comment</td>
</tr>
<tr>
<td>Comment variable (Discussions)</td>
</tr>
<tr>
<td>Comment(Discussion Types)</td>
</tr>
<tr>
<td>Comments</td>
</tr>
<tr>
<td>Compound</td>
</tr>
<tr>
<td>Compound filters</td>
</tr>
<tr>
<td>Compound sort</td>
</tr>
<tr>
<td>Compound statements</td>
</tr>
</tbody>
</table>
Concatenation (attribute base type) 455
Concatenation (attribute definition) 442
Concatenation (attribute) 430
Concatenation (base types) 113
Concatenation (buffers) 164
Concatenation (dates) 147
Concatenation (history type) 332
Conditional statements 20
confAppend 144
confCopyFile 144
confDeleteDirectory 143
confDeleteFile 144
confDownloadFile 147
confFileExists 145
Configuration file access 141
confirm 487
confirmPasswordDialog 226
confMkdir 142
confRead 143
confRenameFile 144
confUploadFile 146
confWrite 143
Constants
and general functions 861
dictionary 862
icons 484
language 861
of type int 755
spell check modes 862
table 817
trigger 779
type bool 116
Constants (history type) 331
Constants for codepages 205
Constrained placement
basics 599
introduction 599
contains 166, 648
containsOle 747
contents 647
Continue statement 23
Controlling
Electronic Signature ACL 351
Rational DOORS from applications that support automation 771
Convert to real 123
convertFolderToProject 281
convertFromCodepage 208
convertProjectToFolder 280
convertToCodepage 208
convertToFolder(ProblemItem) 894
copy(module) 300
copy(partition definition) 706
copyFile 137
copyPassword 226, 236
copyPictureObject 755
copyToClipboard 190
cos(Real Angle) 125
create 815
create(array) 172
create(attribute definition) 442
create(attribute type) 460
create(Baseline Set) 323
create(baseline) 306
create(BaselineSetDefinition) 313
create(buffer) 167
create(descriptive module) 298, 342
create(dialog box) 491
create(Discussion) 901
create(external link) 414
create(folder) 281
create(formal module) 297
create(link module) 298
create(linkset) 410
create(object) 380
create(oleAutoArgs) 769
create(partition definition) 706
create(Project) 285
create(skip list) 154
create(status handle) 181
create(view definition) 679
createButtonBar 492, 637
createChecksumFile 927
createCombo 492, 642
createDropCallback 793
createEditableCombo 590
createItem 492, 637
createMenu 635
createPasswordDialog 224
createPopup 642
createPrivate 679
createPublic 679
createString(skip list) 154
Creating a user account example program 228
Current folder, setting 279
Current module, setting 290
Current object 376
Current object, setting 376
Current page setup, setting 812
Current project, setting 282
current(folder) 279
current(module) 291
current(object) 377
current(page setup) 812
current(project) 283
current(trigger) 786
currentANSICodepage 206
currentColumn(get) 696
currentDirectory 137
currentView 665
Customizing Rational DOORS 193
cutRichText 841

data(for ModuleVersion) 310
Database Explorer 195
Database Integrity Checker 887
Database Integrity Perms 888
Database Integrity Types 887
Database Properties 210
Database properties 213
date 151
date(DBE date_dbe) 534
dateAndTime 152
dateOf 150
dateOf(BaselineSet) 325
dateOnly 152
Dates 147
Dates, comparison 148
DBE resizing 608
DdcMode constants 730
Declarations 15
Declarators 15
decodeResourceURL 427
decodeURL 420
Default link module 410
defaultTableAttribute 831
delete 815
delete(array) 173
delete(attribute definition) 443
delete(attribute type) 461
delete(Baseline) 312
delete(baseline) 306
delete(BaselineSetDefinition) 316
delete(buffer) 167
delete(column) 690
delete(entry) 155
delete(group attribute) 242
delete(IntegrityResultsData&) 895
delete(IPC channel) 190
delete(item in tree view) 505
delete(item) 274
delete(link) 408
delete(linkset) 411
delete(lock list) 936
delete(ModuleProperties) 347
delete(oleAutoArgs) 769
delete(option or item) 505
delete(partition definition) 706
delete(regexp) 161
delete(skip list) 155
delete(status handle) 182
delete(trigger) 783
delete(user attribute) 242
delete(user property) 243
delete(view definition) 680
delete(view) 666
deleteAllMembers 247
deleteCell 820
deleteColumn 547, 820
deleteDiscussion 902
deleteFile 138
deleteGroup 246
deleteKeyRegistry 186
deleteMember 247
deleteNotifyUser 224
deletePicture 756
deleteRow 820
deleteTable 820
deleteUser 246
deleteValueRegistry 187
Derived types 15
descendants(show/hide) 665
descendants(state) 666
descending 661
description 270
description(BaselineSetDefinition) 315
Descriptive modules 341
deselect 387
deselect(archive item) 924
destroy(dialog box) 492
destroy(icon) 485
destroy(Bitmap) 572
destroySort 664
Developing DXL programs 3
Diagnostic perms 856
Dialog boxes
  elements 501
  example program 603, 604
  functions 489
Dictionary
  constants 862
  variable 879
diff(buffer) 337
directory 182
disableDisplayWarnings 859
disableGeneralRichTextWarnings 858
disableObjectTextAssignmentWarnings 857
disableObjectTextRichTextWarnings 857
disableRepeatWarnings 858
disconnect 189
Discussion 897
discussion (trigger) 906
Discussion variable (Discussions) 898
Discussions 897
DiscussionStatus 897
display 698
Display Color Schemes 193
displayRich 699, 700
dispose(Discussion/Comment) 906
dispose(partition definition) 706
Document attributes 809
document(module) 372
doorsInfo 224
doorsVersion 286
downgrade 299
downgradeShare 299
Drag-and-drop 545
draggedObjects 796
draw 568
drawAngle 569
drawBitmap 572
dropDataAvailable 794
droppedAttributeOLEText 796
droppedAttributeRichText 795
droppedAttributeText 795
droppedAttrOLETextAvailable 795
droppedAttrRichTextAvailable 795
droppedAttrTextAvailable 794
droppedList 796
droppedString 794
DXL attribute
  example program 466
  introduction 465
dxl(set) 691
dxl(trigger) 786
dxlHere() function 914
dx1WarningFilename 859
edxWarningLineNumber 859
Dynamic triggers 778

E

echoed
  inlinks 403
  outlinks 403
edit(open module) 300
ElectronicsSignature Data Manipulation 355
ellipse 568
Embedded OLE objects 737
empty(element) 506
enableDefaultTableAttribute 831
enableDisplayWarnings 859
enableGeneralRichTextWarnings 858
enableObjectTextAssignmentWarnings 856
enableObjectTextRichTextWarnings 857
enableRepeatWarnings 858
end(button bar) 643
end(configuration area stream) 145
end(menu) 643
end(of match) 161
end(popup) 643
end(stream) 138
endPrintJob 573
ensureUserRecordLoaded 232
entire(module) 372
error function 912
errorBox 486
Errors 911
escape 130
event(trigger) 785, 793
events
  names 779
  types 779
everSectioned 895
Example (Discussions) 908
Example (HTML Control) 614
Example (HTML Edit Control) 620
Example programs
  add a signature 360
  attribute definitions 448
  baselines 309
  creating a user account example 228
  DXL attribute 466
files and streams 140
filters 655
history 340
list signatures 363
placing dialog boxes 603, 604
progress bar 607
regular expressions 162
RIF 731
setting access control 479
skip lists 157
sorting 664
spelling/dictionary 884
status handle 184
text buffers 170
text views 677
excludeCurrent 648
excludeLeaves 648
excludes 660
exists(attribute definition) 444
exists(tree view) 550
existsGroup 231
existsUser 231
exp(Real x) 125
export 572
exportPackage 725
exportPicture 756, 757
exportRTFString 850
exportType 725
Expressions 18
ExternalLink 413
ExternalLink current 414
ExternalLinkBehavior 414
ExternalLinkDirection 413
extractAfter 343
extractBelow 344

F
field 530
File inclusion 11
fileName 529
Files and streams
  example program 140
  introduction 133
Filter attributes, comparing 646
Filtering on multi-valued attributes 660
Filters
  example program 655
  introduction 645
filterTables 649
find 230
find(attribute definition in ModuleProperties) 347
find(attribute definition) 444
find(attribute type) 455
find(entry) 155
findAttribute 710
findByID 231
findGroupRDNFromName 250
Finding
  links 395
  objects 371
findLinkset 710
findModule 709
findPlainText 129, 674
findRichText 841
findUserInfoFromDN 251
findUserRDNFromName 250
findByUser 710
firstNonSpace 167
flush 136
flushDeletions 383
fnMiniExplorer 198
folder(handle) 280
folder(IntegrityProblem) 891
folder(state) 270
font 564
Font constants 195
fontTable 854
for {string|ModuleVersion} in recentModules 345
for all incoming external links 415
for all outgoing external links 415
for AttrType in ModuleProperties 348
for Comment in Discussion 901
for Discussion in Type 900
for group in ldapGroupsForUser 235
for int in installedCodepages 206
for int in supportedCodepages 206
for Locale in installedLocales 199
for Locale in supportedLocales 200
For loop
  for access record in all Baseline Set Definition 319
  for access record in all type 477
  for access record in Baseline Set Definition 319
  for access record in type 476
  for access record in values 477
  for all incoming links in 396
  for all incoming links in all 401
  for all items in folder 278
for all items in project 278
for all link references in 400
for all modules in project 292
for all outgoing links in 395
for all outgoing links in all 401
for all projects in database 284
for all source link references in 402
for all source links in 402
for all source references in 399
for all sources in 397
for all spellings in alternatives 865
for alternativeWord in Dictionary 881
for archive item in archive 915
for attribute definition in module 447
for attribute type in module 458
for baseline in module module
for baseline Set in BaselineSetDefinition 322
for baselineSet in ModName_ 327
for baselineSetDefinition in Folder 313
for baselineSetDefinition in ModName_ 313
for buffer in Dictionary 880
for buffer in SpellingAlternatives__ 869
for cell in row 821
for columns in module 697
for data element in skip list 157
for each incoming link in 396
for each source 398
for each source reference 399
for file in configuration area 146
for file in directory 139
for group in database 234
for history record in type 335
for history session in module 340
for in-partition in project 721
for IntegrityCheckItem in IntegrityResultsData 889
for IntegrityProblem in IntegrityResultsData 890
for IntegrityProblem in ProblemItem 890
for item in folder 277
for language in Languages__ 871
for link module descriptor in folder 400
for lock in lock list 937
for module attributes in module 434
for module in BaselineSetDefinition 315
for module in database 291
for Module in Folder do 293
for moduleVersion in all BaselineSet 326
for moduleVersion in BaselineSet 326
for object attributes in module 434
for object in all 373
for object in document 374
for object in entire 373
for object in module 374
for object in object 375
for object in top 375
for open module in project 292
for out-partition in project 722
for partition attribute in partition module 714
for partition definition in project 722
for partition module in partition definition 713
for partition view in partition module 714
for pictures in project 765
for position in list (selected items) 544
for position in list view (selected items) 553
for ProblemItem in IntegrityResultsData 889
for project in database 283
for property in user account 241
for rich text in string 836
for row in table 821
for setup name in setups 816
for spellingError in SpellingErrors__ 867
for trigger in database 784
for trigger in module 785
for trigger in project 784
for user in database 233
for user in group 234
for user in notify list 235
for value in list (selected items) 544
for value in list view (selected items) 553
for view in module 673
for module level attribute definition in {Module|ModuleProperties} 448
for RifDefinition in Project 730
for RifImport in RifDefinition 731
for RifModuleDefinition in RifDefinition 730
for sort in sort 664
for string in Fonts__ do 205
for string in longDateFormats 152
for string in ModuleProperties 347
for string in shortDateFormats 151
for View in View 675
forename 237
formalStatus 301
format 139
frame 528
full 598
fullHostname 179
fullName 236
fullName(item) 271
fullName(ModuleVersion) 312
Function calls 19
Function definitions 16
Functions
   backtrace 912
callstack 912
   key 156
match 160
   put 156
Fundamental functions 113
Fundamental types 15, 113
Future HttpRequest 945

G
General language facilities 133
get 471
Get baseline data 306
Get display state 303
Get page dimension 807
Get page properties status 805
get(archive data) 925
get(BaselineSetDefinition) 317
get(data from array) 173
get(DBE date_dbe) 535
get(element or option) 508
get(group attribute) 243
get(HTML frame) 612
get(HTML view) 611
get(selected text) 174
get(user attribute) 243
get(user property) 243
get(view definition) 679
getAccountsDisabled 214
getAdditionalAuthenticationEnabled 263
getAdditionalAuthenticationPrompt 264
getAddressAttribute 258
getAdministratorName 227
getAlternative 881
getArchiveType 918
getAttribute 614
getAttributeFilterSettings_ 649
getBorderSize 493
getBoundedAttr 445
getBoundedUnicode 435
getBuffer 613
getBuffer(DBE date_dbe) 535
getBuffer(DBE) 523
getCachedExternalLinkLifeTime 418
getCachedExternalLinkLifeTime 418
getCanvas 700
getCaptionHeight 493
getCatalanOptions 876
getCellAlignment 825
getCellShowChangeBars 825
getCellShowLinkArrows 825
getCellWidth 825
getCheck 548
columnBottom 385
columnFilterSettings_ 650
columnTop 386
columnValue 548
columnLinePasswordDisabled 264
columnComment(SignatureEntry) 359
columnComments(Discussion d) 903
columnComponentFilter_ 659
columnCompoundFilterType_ 659
columnCorrectionComplete(SpellingError) 869
columnCursorPosition 392
columnDatabaseIdentifier 214
columnDatabaseMailPrefixText 211
columnDatabaseMailServer 219
columnDatabaseMailServerAccount 220
columnDatabaseMinimumPasswordLength 217
columnDatabaseName 213
columnDatabasePasswordRequired 215
columnDate(DBE date_dbe) 535
columnDate(SignatureEntry) 358
columnDateFormat 199
columnDef 471
columnDefaultColorScheme 193
columnDefaultLineSpacing 204
columnDefaultLinkModule 410
columnDefaultViewForModule 669
columnDefaultViewForUser 669
columnDescription 405
columnDescriptionAttribute 257
columnDisableLoginThreshold 221
columnDiscussions 903
columnDoorsBindNameDN 253
columnDoorsGroupGroupDN 255
columnDoorsGroupRoot 254
columnDoorsUserGroupDN 255
columnDoorsUsernameAttribute 256
columnDoorsUserRoot 254
columnDOSstring 167
columnDXMLFileHelp 550
columnDXMLFileName 550
columnEditDXMLControlled 212
getEmail(SignatureEntry) 358
ggetEmailAttribute 257
ggetEnglishOptions 874
ggetenv 178
ggetErrorStartPos(SpellingError) 868
ggetErrorStopPos(SpellingError) 868
ggetErrorString 868
ggetExplanation 872
ggetFailedLoginThreshold 221
gsetFontSetting 204
gsetFontSettings 195
ggetFormattedLocalDate(SignatureEntry) 359
ggetFrenchOptions 875
ggetGermanOptions 875
ggetGrammarLevel 877
ggetGrammarRules 872
ggetGreekOptions 876
ggetGroupMemberAttribute 259
ggetNameAttribute 259
ggetGroupObjectClass 258
ggetHTML 612
ggetId 871
ggetIgnoreReadOnly 879
ggetImplied 471
ggetInnerHTML 613
ggetInvalidCharInModuleName 296
ggetInvalidCharInProjectName 284
ggetInvalidCharInSuffix 307
ggetIsValid(SignatureEntry) 360
g.getLabel(SignatureEntry) 359
ggetLabelOptions(SignatureEntry) 359
ggetLabelSpecifier(SignatureInfo) 356
ggetLanguage 871, 873
ggetLdapServerName 252
ggetLegacyLocale 202
ggetLegacyURL 424
ggetLineSpacing 203
ggetLineSpacing(Locale) 204
ggetLinkFilterSettings_ 650
ggetLocalDate(SignatureEntry) 359
ggetLocksInDatabase 932
ggetLocksInFolder 933
ggetLocksInModule 933
ggetLogicalColorName 595
ggetLoginFailureText 211
ggetLoginLoggingPolicy 222
ggetLoginNameAttribute 256
ggetLoginPolicy 220
ggetMaxClientVersion 223
ggetMaxPasswordAgeLimit 219
ggetMaxPasswordGenerationLimit 218
ggetMemoryUsage 37, 178
ggetMessageOfTheDay 209
ggetMessageOfTheDayOption 210
ggetMinClientVersion 223
ggetMinPasswordAgeInDays 219
ggetMinPasswordGeneration 218
ggetModuleDetails 919
ggetMostRecentBaseline 307
ggetMostRecentBaseline(Module) 312
ggetName 406, 871, 872
ggetNextError 866
ggetObjectByRifID 727
ggetObjectDiscussions 903
ggetObjectFilterSettings_ 650
ggetOleWidthHeight 748
ggetOptions 873
ggetOverridable 407
ggetParent 497
ggetParentFolder(item) 271
ggetParentProject(item) 271
ggetPartitionMask 723
ggetPartitionMaskDef 723
ggetPartitionMaskVal 723
ggetPictBB 758
ggetPictFormat 758
ggetPictName 758
ggetPictWidthHeight 759
ggetPort 188
ggetPortNo 252
ggetPos 492
ggetProjectDetails 920
ggetProperties 347
ggetRealColor 595
ggetRealColorIcon 595
ggetRealColorName 595
ggetRealColorOptionForTypes 456
ggetReconfirmPasswordRequired 215
ggetReconfirmPasswordTimeout 216
ggetReference 273
ggetRegistry 184
ggetRequireLettersInPassword 216
ggetRequireNumberInPassword 216
ggetRequireSymbolInPassword 217
ggetResourceURL 426
ggetResourceURLConfigOptions 427
ggetRifID 727
ggetRow 824
getRussianOptions 877
getSearchObject 388
getSelectedCol 295
getSelection 387
gGetSentenceStartPos(SpellingError) 868
gGetSentenceStopPos(SpellingError) 869
gGetShowTableAcrossModule 826
gGetSignatureInfo(SignatureInfo si&, ModName_ document, int major, int minor, string suffix) 355
gGetSimpleFilterType_ 649
gGetSize 493
gGetSortColumn 549
gGetSource getTarget 411
gGetSourceName 406
gGetSourceVersion(Linkset) 404
gGetSpanishOptions 876
gGetSpellingCheckingMode 878
gGetSpellingFirst 878
gGetSystemLoginConformityRequired 264
gGetTable 824
gGetTargetName 406
gGetTDBindName 262
gGetTDPortNumber 262
gGetTDServerName 261
gGetTDUseDirectoryPasswordPolicy 263
gGetTelephoneAttribute 257
gGetTemplateFileName 551
gGetTimerID 954
gGetTimerInterval 954
gGetTimerName 954
gGetTimerRunning 955
gGetTitle 493
gGetUKOptions 874
gGetURI 457
gGetURL 420, 611
gGetUseLdap() 249
gGetUserFullName 358
gGetUserlistDetails 921
gGetUserName(SignatureEntry) 358
gGetUseTelelogicDirectory 261
gGetVal 471
gGetWord 881
gluedHelp 494
goodFileName 137
goodStringOf 335
gotoObject 377
Grammar Constants 862
graphics(get) 692
graphics(set) 692
Groups and users
management 238
manipulation 230
properties 239, 260
H
halt function 25
hardDelete(module) 301
hardDelete(object) 383
hasFocus 500
hasHeader 579
hasInPlace 575
hasLinks 651
hasNoLinks 652
hasPermission(SignatureInfo, Permission) 352
hasPermission(SignatureInfoSpecifier__, Permission) 352
hasPermission(string, SignatureInfo, Permission) 352
hasPermission(string, SignatureInfoSpecifier__, Permission) 353
hasPicture/exportPicture 700
hasScrollbars 581
hasSpecificValue 446
headerAddColumn 579
headerChange 579
headerRemoveColumn 580
headerReset 580
headerSelect 580
headerSetHighlight 580
headerShow 581
height 565
help 494
hide 629
hide(dialog box) 494
hide(element) 504
hideExplorer 672
Hierarchy
clipboard 267
information 270
manipulation 274
highlightText 389
History
example program 340
introduction 331
properties 333
home 555
Horizontal navigation 379
hostname 179
HTML Control 609
HTML Edit Control 619
HTML functions 938
htmlBuffer 620
htmlEdit 619
htmlText 938
htmlView 609
HTTP Server 940
HttpBody 944
HttpHeader 943
HttpHeaderEntry 944
HttpRequest 940
HttpRequest(error message) 941
HttpRequest(timeout and error message) 941
HttpRequest(timeout) 941
HttpResponse 943
HttpVerb 942

id(Locale) 201
identifier(object get) 386
Identifiers 11, 14
ignoreWord 869
Immediate declaration 16
Importing rich text 855
importPicture 760
importRifFile 726
importRTF 855
inactive 504
inClipboard 269
includeCurrent 651
includeLeaves 651
includes 660
includesTime 152
info(get) 692
info(set) 692
infoBox 486
Information about objects 384, 388
inherited 472
inherited(BaselineSetDefinition) 318
inheritedDef 472
inheritedVal 472
In-partition properties 721
In-place editing 574
inPlaceChoiceAdd 576
inPlaceCopy 576
inPlaceCut 576
inplaceEditing 389
inPlaceGet 576
inPlaceMove 575
inPlacePaste 576
inPlaceReset 577
inPlaceSet 577
inPlaceShow 575
inPlaceTextHeight 577
insert 882
insert(column in module) 692
insert(item in list view) 506
insert(item in tree view) 506
insert(option or item) 506
insertBitmapFromClipboard 760
insertCell 823
insertColumn(list view) 548
insertColumn(table) 823
insertDroppedPicture 797
insertPictureAfter 761
insertPictureBelow 761
insertPictureFile 761
insertPictureFileAfter 762
insertPictureFileBelow 763
insertRow 824
installed(Locale) 201
instance 621
Integer comparison 121
Integer constants 12
IntegrityCheckItem 887
IntegrityItemType 887
IntegrityProblem 887
IntegrityResultsData 887
Interprocess communications 187
intOf(char) 119
intOf(date) 150
intOf(real) 124
Introducing DXL 3
ipcHostname 188
isAccessInherited 472
isAccessInherited(BaselineSetDefinition) 318
isanyBaselineSetOpen(BaselineSetDefinition) 317
isAscending 663
isAttribute(group attribute) 242
isAttribute(user attribute) 241
isAttribute(user) 241
isAttributeValueInRange 445
isBatch function 26
isDatabaseDict 882
isDefault 473
isDefaultURL 426
isDeleted(item) 271
isDeleted(project name) 285
isDescending 663
isDiscussionColumn 904
isEdit 295
isFirstObjectInDXLSet(Object) 701
isinheritedView 667
isLastObjectInDXLSet(Object) 701
isLocked 934
isLockedByUser 478, 934
isLockedClosed 934
isMember 438
isNull 652
isOleObjectSelected 746
isOpen(BaselineSet) 325
isPartitionedOut 723
isPartitionedOutDef 723
isPartitionedOutVal 723
isRanged 455
isRead 295
isRichText 842
isShare 295
isSupported 872
isTimer 954
isUsed 455
isValidChar 208
isValidDescription 296
isValidInt 121
isValidName 285, 296, 667, 815
isValidPrefix 296
isVisible 297
isVisibleAttribute 446
Item access controls 266
item(handle) 277
itemClipboardIsEmpty 269
itemFromID(handle) 277
itemFromReference 273
Iterators (Discussions) 900

L
label 525
Language 870
  and Grammar 870
  constants 861
  fundamentals 7
language(Locale) 200
Languages__ 870
lastError function 913
Layout
  context 698
  DXL 698
layoutDXL 550
LDAP
  configuration 250
  data configuration 256
  server information 252
left 597
leftAligned 597
length 128
length(buffer get) 168
length(buffer set) 168
level modifiers 779
level(module get) 302
level(module set) 302
level(object get) 386
level(trigger) 785, 793
levelModifier 787
levels 779
Lexical conventions 10
Library description file format 625
line 567
Line spacing constant for 1.5 lines 203
link(get) 694
link(set) 694
link(Trigger) 789
linkIndicators(show/hide) 667
linkIndicators(state) 668
Links
  access control 394
  creation 394
  management 404
  operators 394
  source 408
  target 409
linkset 411
list 542
listView 547
load 307, 485, 668

J
justify(get alignment) 693
justify(set alignment) 693

K
key function 156
Keyboard event constants 560
keyword(buffer) 168
kind 787
load(linkset) 412
load(ModuleVersion) 310
load(partition definition) 707
loadBitmap 571
loadChecksumFile 928
loadDirectory 232
loadInPartitionDef 707
loadLdapConfig() 248
loadUserRecord 231
locale 201, 202
Locale type 199
Localizing DXL 6
Lock
    manager 931
    properties 931
lock(BaselineSetDefinition) 316
lock(module) 934
lock(object) 478, 935
Locking 478
log(Real x) 125
Logical colors 591
Loop statements 22
Looping within
    projects 287
Loops 9
lower 128

M
main(get) 693
main(set) 694
major(BaselineSet) 323
markUp 342
match function 160
matches 160
maximumAttributeLength 429
mayEditDXL 236
mayModifyDiscussionStatus 903
member 247
Menu
    DXL file format 626
    index file format 626
menuBar 582
Menus, status bar and tool bars example 583
menuStatus 628
Message boxes 486
messageBox 488
Mini database explorer 198
Minimum and maximum operators 121
minimumSize 30, 500
minor(BaselineSet) 324
Miscellaneous
    object functions 389
    spelling 882
mkdir 179
mode 183
modified 555
modified(date) 182
modify(attribute definition) 446
modify(attribute type) 461
module(containing object) 372
module(handle) 291, 309
module(link) 408
module(state) 270
module(Trigger) 788
ModuleProperties 346
Modules
    access controls 289
    display state 302
    information 293
    manipulation 297
    menus 630
    properties 346
    recently opened 344
    references 290
    state 294
    status bars 627
moduleVersion(handle) 310
move(item) 276
move(object) 382
multiList 543
Multi-value enumerated attributes 437

N
name(BaselineSetDefinition) 314
name(item) 270
name(Locale) 200
name(ModuleVersion) 311
name(trigger) 787
name(view) 668
Naming conventions 9
Navigation from an object 377
next(filtered) 668
nextMajor 308
nextMinor 308
noElems 507
noError function 913
null 114
null constant 14
Null statement 24
number(history session) 336
number(object get) 386

O
object 390
object(absno) 371
object(trigger) 787
Objects
access controls 369
managing 380
status 385
of function 24
ok(arrows) 559
ok(buttons) 556
OLE
information functions 748
objects 737
OLE clipboard 737
oleActivate 737
oleCloseAutoObject 743
oleCopy 739
oleCount 746
oleCreateAutoObject 767
oleCut 740
oleDeactivate 738
oleDelete 741
oleGet 768
oleGetAutoObject 768
oleGetResult 767
oleInsert 741
oleInsert (insert to buffer) 742
oleIsObject 742
oleLoadBitmap 763
oleMethod 771
olePaste 744
olePasteLink 745
olePasteSpecial 744
olePut 769
oleResetSize 752
oleRTF 743
oleSaveBitmap 746
oleSetHeight and Width 752
oleSetMaxLength 751
oleSetMinHeight 751
oleSetResult 767
open(Dictionary) 879
open(partition file) 715
openPictFile 764
openProject 286
Operating system
commands 177
interface 177
Operations on
all types 113
type bool 115
type char 116
type int 119
type real 122
type string 126
Operator functions 17
Operators 469
minimum and maximum 121
template expressions 622
unary 120
opposite 597
Options Constants 861
optionsExist 194
OSLC DXL Services 949
OSLC Link Discovery 418
OSLCDXLService properties 949
Out-partition properties 720
Overloaded functions and operators 19

P
pageBreakLevel(get) 809
pageBreakLevel(set) 809
pageColumns(get) 813
pageColumns(set) 813
pageExpandHF 811
pageFormat(get) 813
pageFormat(set) 813
pageHeaderFooter(get) 810
pageHeaderFooter(set) 810
pageLayout 815
pageTitlePage 814

Pages
dimensions 806
setup information 812
setup management 815
pageTitlePage 814
pageTOCLevel(get) 809
pageTOCLevel(set) 809
parentRef(IntegrityProblem | ProblemItem) 893
parentRefID(IntegrityProblem | ProblemItem) 893
Parse time errors 911, 940
Parsing 8
partition 470
Partitions
   access 722
   attribute properties 719
   definition contents 708
   definition management 705
   definition properties 719
   file properties 720
   information 717
   management 714
   module properties 719
   properties 718
   view properties 719
pasteToEditbox 843
path(item) 271
Persistent triggers 778
Picture object support 755
pictureCompatible 766
pictureCopy 764
platform 177
polarLine 570
polygon 570
pow(Real x) 126
Pragmas 12
preloadView 667
previous(filtered) 668
print 573
print(attribute base type) 456
print(base types) 114
print(date) 149
print(history type) 335
printCharArray 175
printModule 299
priority 789
ProblemItem 887
problems(IntegrityResultsData, string) 891
Progress bar
   example program 607
   introduction 605
progressCancelled 607
progressMessage 606
progressRange 606
progressStart 605
progressStep 606
progressStop 607
project(handle) 283
project(state) 270
Properties 469
Properties (Discussions) 897
purge(item) 275
purgeObject_ 384
purgeObjects_ 384
put function 156
put(data in array) 175
put(oleAutoArgs) 770
putString 175

Q
qualifiedUniqueID 273
query 488

R
radioBox 533
raise 495
random(int) 122
random(real) 126
Range 20
Rational Directory Server 260
Rational DOORS
   built-in windows 628
   customizing 193
   database access 213
Rational DOORS URLs 420
Read
   and write operators 165
   from stream 134, 141
   line from stream 134, 142
read 207
read(BaselineSetDefinition) 317
read(open file) 135
read(open module) 300
readFile 136, 207
ready 495
Real colors 593
realBackground 562
realColor 563
realize(pending) 495
realize(show) 496
realOf 124
recentModules 344
rectangle 566
recv 189
Reference operations 18
refresh 304
refreshDBExplorer 197
refreshExplorer 196
regexp 161
regexp2 162
region(Locale) 201
registeredFormat 794
regular 182
Regular expressions
  example program 162
  introduction 158
reimportPicture 765
reject 653
rejoinPartition 717
rejoinReport 717
release 496
remove 882
remove{lock} 936
removeAttribute 712
removeGroup (Discussions access controls) 907
removeLinkModuleDescriptor 405
removeLinkset 712
removeModule 711
removeModule(BaseLineSetDefinition) 316
removeOSLCDXLService(OSLCDXLService service) 951
removeOSLCDXLService(string key) 951
removePartition 717
removeRecentlyOpenModule(ModuleVersion) 345
removeUnlistedRichText 836
removeUser (Discussions access controls) 907
removeView 712
rename(archive item) 924
rename(BaselineSetDefinition) 314
rename(item) 276
rename(partition definition) 707
renameFile 138
reopenDiscussion 902
repaired(IntegrityProblem) 894
repaired(ProblemItem) 894
replaceRichText 842
Reporting access control example 480
requestLock 938
resetColor 194
resetColors 194
restore(archive) 921
restoreFiles 923
restoreModule 922
restoreProject 923
restoreUserlist 924
Return statement 23
returnPartition 716
Rich text
  constructors 834
  processing 833
  strings 840
tagS 833
richClip 843
richField 530
RichText type properties 838
richtext_identifier(Object) 843
richText(box) 554
richText(column) 835
richText(of attribute) 846
richText(of string) 847
richTextFragment 850
richTextNoOle 849
richTextNoOle(column) 835
RichTextParagraph type properties 837
richTextWithOle 848
richTextWithOle(column) 835
richTextWithOleNoCache 849
richTextWithOleNoCache(column) 835
RIF example programs 731
RIfDefinition 728
RIfImport 726
rifMerge 728
RIfModuleDefinition 729
right 597
row 819
rtfSubString 846
runFile 772
runStr 773
Run-time errors 912
S
save(BaselineSetDefinition) 317
save(module) 300
save(page setup) 816
save(partition definition) 707
save(SignatureInfo si, int &code) 357
save(SpellingOptions) 873
save(view definition) 680
save(view) 670
saveClipboardBitmapToFile 760
saveDirectory 233
saveDiscussionAccessList 908
saveDroppedPicture 797
saveLdapConfig() 248
saveModified(partition definition) 708
saveUserRecord 232
Scope 14
scroll 697
Scrolling functions 697
scrollSet 582
search 170
sectionNeedsSaved 383
select(archive item) 924
select(element) 507
selected(element) 507
selected(item) 508
selectedElems 544
Semicolon and end-of-line 10
send 189
sendBroadcastMessage 951
sendEMailNotification 227, 228
separator(dialog box) 526
separator(menu) 642
server 188
serverMonitorIsOn 180
session 150
set 473
Set display state 304
Set page dimension 807
Set page properties status 806
set(BaselineSetDefinition) 318
set(char in buffer) 169
set(choice element values) 516
set(DBE date_dbe) 535
set=file selector) 517
set(filter) 654
set(html callback) 610
set(HTML edit) 620
set(html URL) 611
set(HTML view) 612
set(icon) 517
set(item value) 516
set(key or mouse callback) 518
set(list view callback) 521
set(select and activate) 520
set(select, deselect, and activate) 522
set(select) 518
set(selected status) 516
set(SignatureInfo, Permission, string name) 353
set(SignatureInfoSpecifier__, Permission, string name) 353
set(sort function) 522
set(sort) 662
set(status bar message) 517
set(tree view expand) 523
set(trigger status) 790
set(user property) 244
set(value or selection) 514
setAccess 713
setAccountsDisabled 214
setAddressAttribute 258
setAllCellsAlignment 826
setAllCellsBorder 826
setAllCellsShowChangeBars 826
setAllCellsShowLinkArrows 827
setAllCellsWidth 827
setAnnotation(BaselineSet) 325
setAttrFromHTML 939
setAttribute 614
setCachedExternalLinkLifeTime 418
setCatalanOptions 877
setCellAlignment 827
setCellBorder 827
setCellShowChangeBars 828
setCellShowLinkArrows 828
setCellWidth 828
setCenteredSize 498
setCheck 549
setColumnAlignment 828
setColumnShowChangeBars 829
setColumnShowLinkArrows 829
setColumnWidth 829
setCommandLinePasswordDisabled 264
setDatabaseMailPrefixText 211
setDatabaseMailServer 219
setDatabaseMailServerAccount 220
setDatabaseMinimumPasswordLength 218
setDatabaseName 213
setDatabasePasswordRequired 215
setDef 473
setDefaultColorScheme 194
setDefaultLinkModule 410
setDefaultViewForModule 670
setDefaultViewForUser 670
setDescription 457
setDescription(BaselineSetDefinition) 314
description(partition definition) 708
descriptionAttribute 257
disableLoginThreshold 221
discussionColumn 904
setDoorsBindNameDN 253
setDoorsBindPassword 253
setDoorsBindPasswordDB 253
setDoorsGroupGroupDN 255
setDoorsGroupRoot 254
setDoorsUserGroupDN 255
setDoorsUsernameAttribute 256
setDoorsUserRoot 254
setDropList 797
setDropString 796
setDXLServiceResult 950
setDXLWindowAsParent 500
setEditDXLControlled 211
setEmailAttribute 257
setempty 169
setEnglishOptions 874
setenv 179
setExtraHeightShare(DBE) 608
setExtraWidthShare(DBE) 608
setFailedLoginThreshold 221
setFocus 495, 523
setFontSettings 196, 205
setFrenchOptions 875
setFromBuffer(DBE date_dbe) 535
setFromBuffer(DBE, Buffer) 524
setGermanOptions 875
setGotFocus 504
setGrammarLevel 878
setGreekOptions 876
setGroup 245
setGroupMemberAttribute 259
setNameAttribute 259
setGroupObjectClass 259
setHTML 612
setIgnoreReadOnly 879
setImplied 474
setInnerHTML 613
setLabelSpecifier(SignatureInfo si, string newLabel) 356
setLanguage 873
setLdapServerName(string) 252
setLegacyLocale 202
setLimits(DBE date_dbe) 534
setLineSpacing 203
setLineSpacing(Locale) 204
setLinkModuleDescriptorsExclusive 405
setLoginFailureText 210
setLoginLoggingPolicy 222
setLoginNameAttribute 256
setLoginPolicy 220
setLostFocus 504
setlower 169
setMaxClientVersion 223
setMaxValue 465
setMessageOfTheDay 209
setMessageOfTheDayOption 209
setMinClientVersion 222
setMinPasswordAgeInDays 219
setMinPasswordGeneration 218
setMinValue 465
setOVERRIDABLE 407
setParent 497
setParent(ProblemItem, Folder) 893
setPortNo 252
setPos 498
setPreloadedView 666
setRealColor 596
setRealColorOptionForTypes 456
setReconfirmPasswordRequired 215
setReconfirmPasswordTimeout 216
setRefreshDelta 702
setRegistry 185
setRequireLettersInPassword 216
setRequireNumberInPassword 217
setRequireSymbolInPassword 217
setRichClip 190, 844
setRichClip(buffer/RTF_string__) 844
setRowWidth 829
setRussianOptions 877
setSearchObject 388
setSelection 387
setServerMonitor 180
setShowDeletedItems(bool) 272
setShowDescriptiveModules 197
setShowFormalModules 197
setShowLinkModules 197
setShowTableAcrossModule 830
setSize 498
setSortColumn 549
setSource 412
setSpanishOptions 876
setSpellingCheckingMode 878
setSpellingFirst 878
setTarget 412
setTDBindName 262
setTDBindPassword 262, 263
setTDPortNumber 47, 262
setTDServerName 261
setTDUseDirectoryPasswordPolicy 263
setTelephoneAttribute 258
setTextChangeCB 510
Setting access control, example program 479
setTitle 499
setUKOptions 874
setUpExtraction 343
setupper 169
setURI 457
setURL 611
setUseLdap() 249
setUser 245
setUseTelelogicDirectory 261
setVal 473
share(open module) 300
shareLock 936
show(dialog box) 496
show(element) 505
show(window) 629
showChangeBars(get) 671
showChangeBars(show/hide) 671
showDeletedModules 304
showDeletedObjects(get) 670
showDeletedObjects(show/hide) 671
showDescriptiveModules(get) 197
showExplorer 672
showFormalModules(get) 197
showGraphicsDatatips(get) 671
showGraphicsDatatips(show/hide) 671
showGraphicsLinks(get) 672
showGraphicsLinks(show/hide) 672
showing 497
showingExplorer 672
showLinkModules(get) 197
showOlePropertiesDialog 747
showPrintDialogs(get) 672
showPrintDialogs(set) 673
side1(module) 412
side2(module) 412
Signature types 351
SignatureInfoSpecifier__ specifier(SignatureInfo) 351
Simple
  elements for dialog boxes 525
  placement 596
sin(Real Angle) 125
Single line spacing constant 203
size 183
sizeof function 24
Skip lists
  example program 157
  introduction 154
slider 531
softDelete(module) 301
softDelete(object) 383
sort function 26
sortDiscussions 902
Sorting
example program 664
modules 661
sorting 663
soundex 128
source 415
sourceAbsNo 409
sourceVersion 403
Specific
  object 377
  windows 629
specific 474
specific(BaselineSetDefinition) 318
specificDef 474
specificVal 474
spell 863, 865
Spell check mode constants 862
spellFix 864
Spelling dictionary 879
Spelling/Dictionary Example programs 884
SpellErrors__ 867
spGetLanguages 870
splitHeadingAndText 391
splitter 526
sqrt(Real x) 126
stacked 598
Standard
  combo box controls 635
  items 630
  menus and submenus 630
  streams 133
start(of match) 161
startConfiguringMenus 499
startPrintJob 573
startTimer 953
Statements
  break 22
  compound 20
  conditional 20
  continue 23
  loop 22
  null 24
  return 23
status 627
Status handle programs example 184
statusBar 582
stopConfiguringMenus 499
stopTimer 953
stored 790, 791
string
buffers 163
buffers example program 170
editor elements 553
text(box) 554
text(column) 695
text(IntegrityCheckItem) 893
Timer 953
timer 953
timestamp(IntegrityCheckItem) 891
title(get) 695
title(set) 695
today 149
toggle 533
toolBar 585
toolBarComboAdd 588
toolBarComboCount 588
toolBarComboBoxCopySelectedText 590
toolBarComboBoxDelete 589
toolBarComboBoxEmpty 588
toolBarComboBoxGetEditBoxSelection 590
toolBarComboBoxGetItem 587
toolBarComboBoxGetSelection 587
toolBarComboBoxInsert 588
toolBarComboBoxPasteText 590
toolBarComboBoxSelect 587
toolBarEditTextGetString 511
toolBarMove 589
Toolbars 585
toolBarShow 589
toolBarVisible 589
top(module) 372
topMost 500
toTable 830
treeView 549
trigger status 790
trigger(dynamic) 782
trigger(persistent) 781
Triggers
constants 779
definition 780
dynamic 778
events 776, 778
introduction 775
level assembly 780
manipulation 783, 793
overview 791
persistent 778
priority 778
scope 777

T

tab 541
Table
  constants 817
  management 818
  manipulation 822
table 819
table(create) 818
tableContents(get) 819
tableContents(set) 820
tan(Real Angle) 125	
targetAbsNo 409
targetVersion 403
tempFileName 137
Template
  expressions 622
  functions 621
template 621	
templates 551
Text
Type bool
  comparison 116
  constants 116
  operations on 115
Type char
  comparison 117
  operations on 116
Type int, operations on 119
Type real
  comparison 123
  operations on 122
  pi 122
Type real constants 13
Type string
  comparison 126
  operations on 126
  substring extraction 127
  type(attribute) 433
  type(IntegrityCheckItem) 891
  type(IntegrityProblem) 892
  type(item) 272
  type(ProblemItem) 892
  type(trigger) 785, 793
Types 15

U
  unApplyFiltering 655
  Unary operators 120
  undelete(item) 275
  undelete(object) 384
  undeleteCell 820
  undeleteColumn 820
  undeleteRow 820
  undeleteTable 820
  undoMarkUp 343
  unicodeString 130, 435
  uniqueID 272
  uniqueID(IntegrityCheckItem) 890
  uniqueID(IntegrityProblem) 890
  uniqueID(ProblemItem) 890
  unixerror function 913
  unload(linkset) 413
  unload(module) 413
  Unlock object functions 479, 937
  unlock(BaselineSetDefinition) 316
  unlock(module) 935
  unset 475
  unset(BaselineSetDefinition) 319
  unset(SignatureInfo, string name) 354
  unsetAll 475
  unsetAll(BaselineSetDefinition) 319
  unsetAll(SignatureInfo) 354
  unsetAll(SignatureInfoSpecifier__) 354
  unsetDef 475
  unsetVal 475
  updateGroupList() 249
  updateToolBarBars 587, 628
  updateUserList() 249
  upper 128
  useAncestors(get and set) 680
  useAutoIndentation 687
  useColumns(get and set) 682
  useCompression(get and set) 684
  useCurrent(get and set) 681
  useDefaultTableAttribute 831
  useDescendants(get and set) 681
  useFiltering(get and set) 685
  useFilterTables(get and set) 682
  useGraphics(get and set) 683
  useGraphicsColumn(get and set) 682
  useLevel(get and set) 684
  useOutlining(get and set) 683
  user 183
  User class constants 238
  user(BaselineSet) 324
  userLocale 200
  username 180, 476
  useRTFColour 524
  useSelection(get and set) 681
  useShowDeleted(get and set) 685
  useShowExplorer(get and set) 683
  useShowLinkIndicators(get and set) 686
  useShowLinks(get and set) 686
  useShowPictures(get and set) 686
  useShowTables(get and set) 686
  useSorting(get and set) 684
  useTooltipColumn(get and set) 684
  useWindows(get and set) 687

V
  validateDOORSURL 425
  value 791
  value(Trigger) 789
  Variables 14
  version 294
  version(Trigger) 789
  Versioned links 401
versionID(BaselineSet) 324
versionString(ModuleVersion) 312
Vertical navigation 378
view 666
Views
  access controls 676
  definitions 678
  elements 545
  example program 677

W
warn function 914
warningBox 487
when(history session) 336
who(history session) 336, 339
width 565
width(get) 696
width(set) 696
wildcard 883
window 628
Windows registry 184
write 207
Write to stream 135, 142
write(open file) 135