

Lotus software



Lotus QuickPlace 3.0



Developer's Guide

Disclaimer

THIS DOCUMENTATION IS PROVIDED FOR REFERENCE PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS DOCUMENTATION, THIS DOCUMENTATION IS PROVIDED "AS IS" WITHOUT ANY WARRANTY WHATSOEVER AND TO THE MAXIMUM EXTENT PERMITTED, IBM DISCLAIMS ALL IMPLIED WARRANTIES, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SAME. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, DIRECT, INDIRECT, CONSEQUENTIAL OR INCIDENTAL DAMAGES, ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS DOCUMENTATION OR ANY OTHER DOCUMENTATION. NOTWITHSTANDING ANYTHING TO THE CONTRARY, NOTHING CONTAINED IN THIS DOCUMENTATION OR ANY OTHER DOCUMENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT GOVERNING THE USE OF THIS SOFTWARE.

Copyright

Under the copyright laws, neither the documentation nor the software may be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form, in whole or in part, without the prior written consent of IBM, except in the manner described in the documentation or the applicable licensing agreement governing the use of the software.

©Copyright IBM Corporation 1985, 2002

All rights reserved.

Lotus Software
IBM Software Group
One Rogers Street
Cambridge, MA 02142

US Government Users Restricted Rights - Use, duplication, or disclosure restricted by GS ADP Schedule Contract with IBM Corp.

List of Trademarks

Domino, Domino Designer, LearningSpace, Lotus, Lotus Notes, LotusScript, Notes, QuickPlace, and Sametime are trademarks or registered trademarks of Lotus Development Corporation and/or IBM Corporation in the United States, other countries, or both. AIX, IBM, iSeries, and OS/400 are registered trademarks of International Business Machines Corporation in the United States, other countries, or both. ActiveX, Microsoft, Outlook, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the United States, other countries, or both. UNIX is a registered trademark of The Open Group in the United States and other countries. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States, other countries, or both.

All other trademarks are the property of their respective owners.

Contents

1 What's New in this Release?	1-1	6 PlaceBots: Automating Tasks in a Place	6-1
What's New in this Release?	1-1	PlaceBots: automating tasks in a place	6-1
QuickPlace API	1-1	Example	6-2
New customizable skin components	1-1	Example	6-4
2 QuickPlace Development Overview	2-1	To run a PlaceBot when a page is submitted	6-6
QuickPlace development overview	2-1	To run a PlaceBot at a scheduled time	6-6
3 QuickPlace Architecture Overview	3-1	To run a PlaceBot manually	6-6
QuickPlace architecture overview	3-1	7 Customizing the Look of a Place	7-1
The QuickPlace file directory structure	3-2	Customizing the look of a place	7-1
4 Customizing Places with the Notes Client and Domino Designer	4-1	Customizable components	7-2
Customizing places with the Notes Client and Domino Designer	4-1	To generate layout files	7-4
To customize existing objects in Notes	4-1	To edit a custom theme	7-5
To create new objects in Notes	4-2	To delete a custom theme	7-5
5 Accessing the QuickPlace API	5-1	To reuse a custom theme	7-5
Accessing the QuickPlace API	5-1	Component name	7-6
XML details	5-7	Usage	7-7
How the XML is processed	5-7	Attribute types	7-8
Specifying the local server	5-8	Usage	7-8
XML example with output	5-9	8 PlaceTypes: Using a Place as a Design Template	8-1
Object APIs	5-10	PlaceTypes: using a place as a template	8-1
Full-text query syntax	5-16	To allow a place to be a PlaceType	8-1
		Index	Index-1

Chapter 1

What's New in this Release?

This chapter describes new features in the IBM® Lotus® QuickPlace™ 3.0 release.

What's New in this Release?

The following are new development features available in IBM Lotus QuickPlace 3.0.

QuickPlace API

By using XML to access the QuickPlace API, you can perform a number of actions in your QuickPlace service, such as searching all places in the service, getting a list of all places on a particular server, creating a new place, or adding a new person or group to a place.

New customizable skin components

There are several new features with skin components you can customize, such as Sametime status, a Room Map link, and a Download File link.

Chapter 2

QuickPlace Development Overview

This chapter is an overview of QuickPlace.

QuickPlace development overview

QuickPlace is a Web tool for team collaboration. It enables the instant creation of a team workspace on the Web. A team can use a place to share and organize ideas, content and tasks around any project. The QuickPlace Developer's Guide contains the following information:

- QuickPlace architecture overview
- Customizing a place with the Lotus Notes® client and Domino Designer®
- Using XML to perform actions in a QuickPlace service
- PlaceBots: automating tasks in a place
- Customizing the look of a place
- PlaceTypes: using a place as a design template
- Sample application: using your own authentication scheme

Chapter 3

QuickPlace Architecture Overview

This chapter describes the QuickPlace design.

QuickPlace architecture overview

Before you customize a place, you should have a basic understanding of the QuickPlace architecture. Although QuickPlace has its own metaphors and object model independent of IBM Lotus Domino™, it is implemented using core Domino technology and takes advantage of Domino data structures. A place is created using templates to structure data, and databases to store the data. Information in a place is stored in data notes — the basic unit of information in a Notes database. The structure of a place is further defined with objects such as rooms, folders, and pages that map to Domino objects.

Because the place objects are based on Domino objects, you can use the Notes client and Domino Designer to view, customize, and create new objects in a place. For more information, see the “Customizing places with the Notes Client and Domino Designer” chapter.

QuickPlace also uses a subset of the Domino/Notes security and authentication model to manage access to a place. It is helpful if you are familiar with the Notes security model, in particular with basic access control list (ACL) settings, and the use of Reader and Author fields. For up-to-date information on Notes application security, see the latest Domino Designer 6 Help, available on the Web at www.lotus.com/ldd/doc.

The following table shows the relationship between QuickPlace objects and Domino objects.

<i>QuickPlace Object</i>	<i>Domino Object</i>	<i>Description</i>
Place	File System Directory	Organizes pages in rooms and folders.
PlaceType	Database template (.ntf)	The structure and design used to create a particular type of place. For example, the default place is Main.nsf, which is created from the MeetingRoom.ntf template. MeetingRoom.ntf is a PlaceType.

continued

<i>QuickPlace Object</i>	<i>Domino Object</i>	<i>Description</i>
Room	Domino Database	A collection of pages with its own security and authentication protection.
RoomType	Database template (.ntf)	The structure and design used to create a particular type of room.
Folder	Domino Folder or View	An organizing structure for collecting and displaying related pages in a site.
Page	Domino Form + Subform + Data Note	The basic vehicle for content. You can create content using the QuickPlace editor or import content from an external source.
Member	Domino Data Note	A member note contains information about a team member of a place. In addition to this data, the member must be listed in the ACL of main.nsf and in a group in names.nsf to pass authentication.
Form	Data note of type "h_Form"	Manages the display of data notes. A form can contain fields for containing data and employ scripts to process and compute data.
Field	Data note of type "h_Field"	Allow for user input of data into data notes.

The QuickPlace file directory structure

QuickPlace data is stored within a subdirectory named QuickPlace, below the Domino server's data directory. The complete directory structure is as follows.

<i>Subdirectory</i>	<i>Content</i>
\Data\QuickPlace\AreaTypes	Contains the templates used to create places and rooms.
\Data\QuickPlace\QuickPlace	Contains Administrator's place files. All places are created from the Administrator's place.
\Data\QuickPlace\<place>	Contains the files of a particular place. <place> represents the name of the place.

Creating places with templates and databases

When you create a place, you are actually creating several Notes databases (NSF files). Databases are created from Notes templates (NTF files).

Templates are like blueprints for databases. Templates contain the forms and fields that define a database built from that template. These forms and fields define the look of the database and how the database processes and stores data. Templates allow for a controlled development environment.

A developer can change a template, then push these changes out to any databases on the server that were created from that template. In QuickPlace, a template is called a PlaceType.

Databases are the basic building blocks of any place.

For more information on Notes templates and databases, see the latest Domino Designer 6 Help, available on the Web at <http://www.lotus.com/ldd/doc>.

Place databases

The following databases are the building blocks of any place:

- The place database — The place database is the parent database in any place. All other databases in the place are children of the place database.

For example, the place database that installs with QuickPlace is Main.nsf. It is created from the PlaceType MeetingRoom.ntf. Main.nsf contains the structure for a group discussion, including a Welcome page, a folder for threaded discussion topics, and the tools for specifying team members and securing the site.

- The Members Directory database — Each place has a Members Directory database. The Members Directory database, Contacts1.nsf, is created from the Members Directory PlaceType Contacts.ntf. The Members Directory database contains all of the data on place members and what access levels they have.
- A room database — A room database structures the contents of a particular room in a place. The default room PlaceType is PageLibrary.ntf, which provides indexing infrastructure for maintaining the pages in a room. This PlaceType also provides security and authentication features so that access to a room can be limited to a subset of team members.

The database created from the PageLibrary PlaceType is assigned a unique name by the system to allow for multiple rooms within a place.

QuickPlace administration templates

When an administrator signs in to a QuickPlace server, they are actually using a place to administer and secure the server. The administrator's place is created from the templates CreateHaiku.ntf and Admin.ntf.

The HaikuCommonForms.ntf template is a central repository for forms used by other templates. This reduces the overhead in a QuickPlace service, allowing for smaller databases, faster creation of a place, and better performance resulting from more efficient server caching.

Chapter 4

Customizing Places with the Notes Client and Domino Designer

This chapter describes how to customize existing QuickPlace objects, and create new objects, using IBM Lotus Notes and IBM Lotus Domino Designer.

Customizing places with the Notes Client and Domino Designer

Because QuickPlace objects are based on Domino objects, you can use the Notes client and Domino Designer to view, customize, and create new objects in a place.

All the data for objects in a place are contained in database notes. To view a place's objects, change an existing object, or create a new object, you can open the place in a Notes client and customize the notes.

You can automate the customization process with Domino Designer. For example, you can write an agent that creates member documents, filling in all of the required fields for identifying members and specifying access to a place.

Note The QuickPlace data schema is subject to change in future versions of the QuickPlace product. Applications written to this data schema may need to be modified in order to work with future versions of the product.

To customize existing objects in Notes

1. Open the place database in Notes.
2. Select the QDK view.
3. Open the object's data note document.
4. Change any values in the document and save it.

Caution Place objects are often interdependent. Changing the value of an object may affect other objects in the place.

To create new objects in Notes

To create a new place object using a form:

1. Open the place database in Notes.
2. Select the QDK view.
3. Create a document that corresponds to the object you are creating. For example, create a member document to create a place member.
4. Fill in field values to define the new object.
5. Save the document to create the object.

Chapter 5

Accessing the QuickPlace API

This chapter describes how to access the QuickPlace API using XML.

Accessing the QuickPlace API

By using XML to access the QuickPlace API, you can perform a number of actions in your QuickPlace service, such as searching all places in the service, getting a list of all places on a particular server, creating a new place, or adding a new person or group to a place.

To access the QuickPlace API, you must set up the QuickPlace server, create well-formed XML that specifies the QuickPlace objects and the actions you want performed on them, then run your XML against the QuickPlace processor on the server. The QPAPI process() method processes the XML, performs the actions, and outputs XML detailing the results.

There are three ways to access the QuickPlace API: You can create an XML input file and run that file against the QuickPlace processor from the command line or Domino server console, write a Java™ program that creates the XML and passes it to the QuickPlace processor programmatically, or use the QPTool execute command. For more information, see the following topics:

- Setting up the QuickPlace server to access the API
- Accessing the API from the command line
- Accessing the API from a Java program
- Accessing the API using a QPTool command
- XML details

Setting up the QuickPlace server to access the API

To access the QuickPlace API you must do the following on the QuickPlace server:

1. Install the Java Developer's Kit (JDK), version 1.1.8. This is the only version supported. You can find the JDK on Sun Microsystems Java Web site at java.sun.com.

2. Add the following files to your CLASSPATH. All are installed with the QuickPlace server:
 - log4j-118compat.jar
 - quickplace.jar
 - xalan.jar
 - xercesImpl.jar
 - xml-apis.jar
3. Add the following to your PATH:
 - Domino server program directory (for example, c:\Lotus\Domino\...)
 - Java JDK bin directory

Note QuickPlace for IBM iSeries™ developers should refer to the “Installing and Managing QuickPlace for iSeries” documentation for instructions on how to setup the server environment to use QuickPlace API. It is available on the Web at <http://www.ibm.com/servers/eserver/series/quickplace>.

Accessing the API from the command line

You can access the QuickPlace API by creating a well-formed XML document and running it against the QuickPlace processor from the command line or Domino server console.

To create and run the XML

1. Create a new XML document in any text editor.
2. Add the appropriate well-formed XML using your own values and save the document as an XML file. For details of the XML needed for specific actions, see this chapter.
3. On the command line, navigate to the /Lotus/Domino directory and enter the following command to execute the XML:

```
java com.lotus.quickplace.api.QPAPI -i inputfile.xml
```

This command is processed by the QuickPlace processor, and the QPAPI.process() method is invoked to process the XML. The JAVA runtime environment you installed is responsible for executing the QPAPI class. You can use the following arguments in your command:

- -i *inputfile.xml* — *inputfile.xml* represents the name of the well-formed XML document that you wish to process.

- `-o outputfile.xml` — `outputfile.xml` represents the name of the XML output document created by QuickPlace which contains all of the processed output and status codes. If you do not specify an output file, XML is output to stdout by default.
- `-session file.xml` — `file.xml` represents the name of the well-formed XML document used by QuickPlace during a search to determine the identity of the user performing the search. This document is required as input when accessing the API to search the QuickPlace service.

For example, to search for the word “SSL” in the service, you must specify the user who is performing the search. For example, “`cn=John Smith,o=IBM.`” The search returns documents that John Smith has access to that contain the word “SSL.” For information on the XML syntax required in this document, see the topic “search (service)” in this chapter.

Note If you run the command from the Domino server console you must preface it with “load.” For example:

```
load java com.lotus.quickplace.api.QPAPI -i inputfile.xml
```

Accessing the API from a Java program

You can access the QuickPlace API by programmatically generating XML and running it against the QPAPI class. The following is required to execute XML from within a Java program:

- You must call the `QPAPI.init()` method prior to any other QuickPlace processing.
- You must either create an XML DOM tree or XML document(s) specifying the QuickPlace API objects and the actions you want taken on them. (If you create XML document(s) they are parsed by the Apache Xerces parser and a XML DOM tree is created.) For more information on creating a XML DOM tree for the QuickPlace API, see the Xerces documentation on the Web at xml.apache.org
- You must call some variation of the `QPAPI.process()` method to initiate the processing of the either an XML DOM tree or XML document(s). See documentation below for more information on the variations of `QPAPI.process()`.
- You must call the `QPAPI.term()` method to terminate the QuickPlace processing and free up resources.

Sample Java code

The following sample Java code fragment illustrates the necessary pieces needed to process the XML document:

```
import com.lotus.quickplace.api.QPAPI;

class TestQPAPI
{
    static void run( )
    {
        // this is called once for the process at startup
        QPAPI.init();
        // build and get XML DOM tree for session and XML action
        script.
        // The programmer provides the functionality to build the
        XML properly in the methods below
        Node sessionXML = buildQPSessionXML();
        Node root = buildQPXML();
        // call this entry point to process using the server
        session
        QPAPI.process( root);
        // call this entry point to process using a user session
        (only used when doing search places)
        // all other actions ignore the session.
        QPAPI.process( sessionXML, root);
        // this is called once for the process at shutdown
        QPAPI.term();
    }
}
```

QPAPI.process()

Whether or not you are creating a Java program, the entry point you use to execute your XML is:

```
QPAPI.process()
```

The `QPAPI.process()` method is responsible for parsing and processing your XML document and executing the supported QuickPlace API actions it encounters. The `QPAPI.process()` method can be called from multiple threads, simultaneously. The `QPAPI.process()` method modifies the input XML DOM tree by modifying, adding, and deleting the necessary nodes.

There are several variations of the `QPAPI.process()` method that you can use:

`process(String sessionFilename, String inFilename, String outFilename)`

This entry point is called by `QPAPI.main()`. So if you execute the `QPAPI` Java class with arguments, this is the entry point that is called. The parameters have the following meanings:

- `sessionFilename` — The pathname of the XML document used by QuickPlace during a search to determine the distinguished name of the user performing the search. This document is required as input when accessing the API to search the QuickPlace service. The document should be properly constructed using the rules and entities specified in the topic “search (service)” in this chapter.
- `inFilename` — The pathname of the XML document to be processed. The document should be properly constructed using the rules and entities specified in this chapter.
- `outFilename` — The pathname of a document to be created by the QuickPlace API that will contain the processed output XML. This document will be very similar to the input document but will be modified by the QuickPlace API to show Action Status Codes and output results of the specific actions, if applicable.

`process(String inFilename)`

This entry point is used when you are not concerned about the output XML. If you use this method, you will not be notified on processing results or error status because they are formatted in the output file. The parameters have the following meanings:

`inFilename` — The pathname of the XML document to be processed. The document should be properly constructed using the rules and entities specified in this chapter.

`process(String inFilename, String outFilename)`

This entry point is used when you want to supply an XML input file and want all processed output to be captured in an XML output file created by the QuickPlace API. The parameters have the following meanings:

- `inFilename` — The pathname of the XML document to be processed. The document should be properly constructed using the rules and entities specified in this chapter.

- `outFilename` — The pathname of a file to be created by the QuickPlace API that will contain the processed output XML. This document will be very similar to the input document but will be modified by the QuickPlace API to show Action Status Codes and output results of the specific actions if applicable.

process(Node session, Node root)

This entry point is used when you have an XML DOM tree. It is primarily used in Java programs that build the XML structure on the fly. The parameters have the following meanings:

- `session` — The session Node. Normally, the session of the server is used when processing the XML document. This is currently only required when executing the XML search API actions. The session node specifies the distinguished name of the user performing the search.
- `root` — The root node of your XML DOM tree object. The root node is usually obtained from an XML parser in a Java application.

process(Node root)

This entry point is used when you have an XML DOM tree. It is primarily used in Java programs that build the XML structure on the fly. The parameters have the following meanings:

`root` — The root node of your XML DOM tree object. The root node is usually obtained from an XML parser in a Java application.

Accessing the API using a QPTool command

QPTool is a server task that you can run with arguments to do administrative tasks. You can use the QPtool execute command to process your XML. To execute an XML file using QPTool, enter the following in either the command line or Domino server console.

When entering in command line:

```
qptool execute arguments
```

When entering in server console:

```
load qptool execute arguments
```

The following table describes the arguments.

<i>Argument</i>	<i>Description</i>
-?	Prints help on the command
-i inputfile	Specifies the XML API file to execute. If no path specified, the default location of the file is the Domino program directory.
-o outputfile	Logs results to a specified XML output file. By default logs results to qptool.execute.xml in the Domino program directory.

For more information on QPTool commands, see the “QuickPlace Administrator’s Guide” on the Web at www.lotus.com/idd/doc.

XML details

The XML you use to access the QuickPlace API consists of elements that represent QuickPlace objects, such as a service, servers, places, people, and groups. The XML for each QuickPlace object has an action attribute associated with it. This action attribute represents the API to be invoked.

How the XML is processed

If you create an XML document, the QuickPlace XML processor processes the XML in the order it appears in the document. When the XML is processed, it is converted into an XML DOM tree structure.

The processor travels down the leftmost branch of the tree, reading each node along the way. If a node contains an action, the processor performs the action, then continues. When it reaches the end of a branch, the processor returns back up the branch to the last node where there was a split and moves down the untravelled branch.

As long as an action is successfully completed, the processor continues on its course. However, if there is an error and the action cannot be completed, the processor stops, backs up to the next sibling node, and continues down the branch.

When the processor has read the entire file, it outputs XML (either in stdout or in another text file) with the results. The output XML is an edited version of the input XML. If an action was completed successfully, the processor removes the action attribute from the XML. The action attributes are removed when successful so that the problems in the output XML can be fixed and the XML used again.

If the action caused an error and was unsuccessful, the action attribute is not removed from the XML. Instead, a <status> and a <message> element are added. The status element contains a number value related to the type of error. The message element contains a text string describing the error. The text string is pulled from the server, and is always in the server’s language.

Note All XML must be well-formed, and must start with at least `<?xml version="1.0"?>` as a processing instruction.

Specifying the local server

XML actions only run on the local server. However, you can use the same XML file on different servers and specify which server a particular action is meant to run on. There are two ways to specify which server an action should run on. You can specify that the action should be run on the local server by adding a `local="true"` attribute to the `<server>` element. For example, `<server local="true">`. Or you can specify the hostname of the local server. For example, `<server><hostname>local server's DNS name</hostname>`.

The following example uses the `<hostname>` element to specify the server each action should run on. The XML below runs the “remove” action on two different places on two different servers. The first action marks *MyPlace1* for removal from *server1*, and the second action marks *MyPlace2* for removal from *server2*. When the XML runs on *server1*, only the *server1* action is processed; when the XML runs on *server2*, only the *server2* action is processed.

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <hostname>server1.enterprise.com</hostname>
      <places>
        <place action="remove">
          <name>MyPlace1</name>
        </place>
      </places>
    </server>
    <server>
      <hostname>server2.enterprise.com</hostname>
      <places>
        <place action="remove">
          <name>MyPlace2</name>
        </place>
      </places>
    </server>
  </servers>
</service>
```

For more information, see the topic “The service node” later in this chapter.

XML example with output

This is an example of properly constructed XML for marking the place *MyPlace* for removal from the local server:

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <places>
        <place action="remove">
          <name>MyPlace</name>
        </place>
      </places>
    </server>
  </servers>
</service>
```

After the remove action is invoked and the action successfully performed, the following XML is returned as output:

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <places>
        <place>
          <name>MyPlace</name>
          <action_result action="remove">
            <status>0</status>
          </action_result>
        </place>
      </places>
    </server>
  </servers>
</service>
```

Object APIs

Click any of the following actions for XML details.

- The service node
- The server node
- The place node
- The placetype node
- The person node
- The group node
- The member node

The service node

The <service> node represents a container for one or more servers that make up the QuickPlace service. Servers that are part of a service can be manipulated with certain actions, such as a search of all places on all servers in the service.

Hierarchy

```
<?xml version="1.0"?>
<service>
  ....
</service>
```

Supported actions

The <service> node supports the following named actions:

- query
- search

query (service)

The query action searches the Place Catalog to find places that a specified person is a member of. The action returns a list of places of which the specified person is a member.

For the query action to work, the Place Catalog must be configured in your service. For information on configuring a Place Catalog, see the “QuickPlace Administrator’s Guide” on the Web at www.lotus.com/ldd/doc.

Syntax

```
<?xml version="1.0"?>
<service action="query">
  <query type="get_member_places">
```

```

    <members>
      <person>
        <dn>distinguished name of person</dn>
      </person>
    </members>
  </query>
</service>

```

Supported attributes

The <query> node supports the following attributes:

type

Values: `get_member_places` — Given a member name, retrieves all places in the service of which the specified name is a member. Member places are listed by server name. Server names are listed by service, which means all servers listed in the Place Catalog.

Results

The results of the search are updated in the XML input tree. The <servers> node is added as a child to the <service> node. For each of the above query types, the results of the query are returned in the following format:

```

<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <name>server1</name>
      <place>
        <name>place1</name>
      </place>
    </server>
    <server>
      <name>server2</name>
      <place>
        <name>place2</name>
      </place>
    </server>
  </servers>
</service>

```

search (service)

The search action performs a full-text search on all places on all QuickPlace servers in the QuickPlace service.

To search all places in the QuickPlace service, you must create a Domain Index on a Domino server in the domain, install a QuickPlace server on the same machine, and configure the Search Places feature on that machine. For information on creating a Domain Index, see the “Domino Administrator Help.” For information on installing a QuickPlace server, see the “QuickPlace Installation and Upgrade Guide.” For information on configuring Search Places on the local server, see the “QuickPlace Administrator’s Guide.” You can find all of this documentation on the Web at www.lotus.com/ldd/doc.

To access the search API, two input files are required: one specifying the search query, and another specifying the distinguished name of the user performing the search. The first file is run using the `-i` argument, the second is run using the `-session` argument. For example, you can create an input file specifying the query called `input.xml`, and another file specifying the user called `session.xml`. To run the search, enter the following command on the command line:

```
java com.lotus.quickplace.api.QPAPI -i input.xml -session
session.xml
```

An error action status is returned if the local server’s QuickPlace configuration specifies that the Search Places feature is disabled, or if Search Places for Anonymous Users is disabled and the user performing the search is anonymous.

You can also access the search functionality through the `QPAPI.process(String sessionFileName, String inFileName, String outFileName)` method via a JAVA program. For more information on accessing the API from a Java program, see the topic “Accessing the API from a Java program” in this chapter.

QuickPlace API actions are always performed on the local server (the server executing the XML). Therefore, in order to perform a domain search, the search action must be run on the server that is configured for Domain Search and contains the Domain Index.

The session file is needed to provide the identity of the user executing the search so that only documents the user has access to are returned.

Syntax for the input file

```
<?xml version="1.0"?>
<service action="search">
  <query order="score | asc | desc" start="n" count="n">
    <![CDATA[Properly formatted full-text query string]]>
  </query>
</service>
```

Note For information on full-text query syntax, see the topic “Full-text query syntax” in this chapter.

Required syntax for the input file

```
<?xml version="1.0"?>
<service action="search">
  <query order="score | asc | desc">
    <![CDATA[Properly formatted full-text query string]]>
  </query>
</service>
```

Syntax for the session file

```
<?xml version="1.0"?>
<service>
  <session>
    <person>
      <dn>distinguished name of user performing search</dn>
    </person>
  </session>
</service>
```

Supported elements for the input file

<query>

Attributes:

- **order** — A value that specifies the search return sort order. The following values are supported:
 - score — Return results sorted by relevance
 - asc — Return results sorted with oldest occurrences first
 - desc — Return results sorted with newest occurrences first

- **start** — Specifies the starting position of the search results to be returned. A value of zero is used if this attribute is not specified. A starting position of zero specifies that results starting with the first match should be returned.
- **count** — Specifies the maximum number of search match hits to be returned. A value of 15 is used if this attribute is not specified. A value of -1 specifies that all hits should be returned.

Example Query:

```
<?xml version="1.0"?>
<service action="search">
  <query start="0" count="100" order="score">
    <![CDATA["quickplace"]]>
  </query>
</service>
```

Supported elements for the session file

<session>

Session represents the connection to the server. Because Search Places only returns results according to the user's access to places, rooms, and documents when searching for a document, Search Places must know who is requesting the search. This information is contained in the session node.

<person>

Specifies the distinguished name of the user performing the search. For example:

```
<dn>CN=Jane Doe,OU=Sales,O=ACME</dn>
```

Results

Below is an example of the XML returned by performing a search for the word: QuickPlace

```
<?xml version="1.0"?>
<service>
  <search_results>
    <search_result seqnum="1">
      <document>
        <title>
          <![CDATA[Release 3.0 Features]]>
        </title>
```

```

    <author local="false">
      <dn>CN=Jane Doe,OU=Sales,O=ACME</dn>
      <name>Jane Doe</name>
    </author>
    <url>https://acmeteam.acme.com:443/QuickPlace/acmeteam/PageLi
brary85256AAF005EC7BB.nsf/1E24BC021C381AE985256AB8004E035B
/4CB455BB81C721AD85256C1300636F10/?OpenDocument</url>
    <abstract>
      <![CDATA[ This document describes the features that are new in
QuickPlace Release 3.0]]>
    </abstract>
    <last_modified>20020812T140737,57-04</last_modified>
  </document>
  <place>
    <name>ACMETeam</name>
  </place>
  <relevance>100</relevance>
</search_result>

```

```

<search_result seqnum="2">
  <document>
    <title>
      <![CDATA[Release 3 Sales Forecast]]>
    </title>
    <author local="false">
      <dn>CN=John Swift,OU=Sales,O=ACME</dn>
      <name>John Swift</name>
    </author>
    <url>https://acmeteam.acme.com:443/QuickPlace/acmeteam/PageLi
brary85256AAF005EC7BB.nsf/h_Index/09A910C51A6818DA85256C0F
00829ADD/?OpenDocument</url>
    <abstract>

```

```

    <![CDATA[ QuickPlace 3 Sales Forecast &nbsp; Sales Staff: Please
    review this document for accuracy and make edits and corrections as
    necessary. This document is used by the ACME Global Sales staff to
    determine the impact of QuickPlace Release 3 with respect]]>
    </abstract>
    <last_modified>20020812T140521,04-04</last_modified>
</document>
<place>
  <name>ACMETeam</name>
</place>
<relevance>100</relevance>
</search_result>
</search_results>
<action_status action="search">
  <code>0</code>
</action_status>
</service>

```

Full-text query syntax

You can find information in a domain by forming queries with search operators. Search operators are words and characters which Domino reads as instructions to search for combinations of words, fields, dates and numbers. It works the same way most Web search engines do (based on Boolean logic), with some very powerful enhancements. For example, you can not only search for two words which appear in the same document, but specify how close they should be to each other, what field they must be in, by their exact case, and that one should be judged as more important. Using wildcards you can also search on just a fragment of a word and Domino returns every word containing that fragment.

Operators are reserved words in Domino. If you want to search for an operator as you would normal text, for example in a phrase such as "Gene and Joan," you must put the phrase in quotes.

<i>Operator</i>	<i>Description and examples</i>
field FIELD [<i>fieldname</i>] (brackets)	These mean “search this field.” Domino then expects you to specify the field to search. In this release of QuickPlace field operators only to find text in the \$Updatedby and _RevisionDate fields. There should be spaces between “FIELD” and words surrounding it. Example: “FIELD \$Updatedby CONTAINS Simpson” finds documents whose \$Updatedby field contains the word Simpson.
() [<i>parentheses</i>]	These determine the order in which Domino processes sections of your query. A part of the query enclosed in parentheses will be processed before parts outside the parentheses.
and AND &	These find documents containing all the conditions or words linked by AND. Example: “cat AND dog AND fish” finds documents containing all three of these words.
or OR ACCRUE , (comma)	These find documents containing either of the conditions or words and returns them ranked by number of appearances in the document.
NOT not !	These make the query negative. You can put NOT between words: “cat AND NOT dog” finds documents containing the word cat, but not the word dog. You can put NOT before any field name: “NOT[author] CONTAINS Simpson” finds documents whose author field does not contain the word Simpson. You can use NOT after CONTAINS, and before a word: “[author] CONTAINS NOT Simpson” finds documents whose author field does not contain the word Simpson. You cannot put NOT after =, <, >, <=, or >= and before a date or number: '[date1] = NOT 12/25/98' does not work.
“ ”	Placing quotes around operators (like AND, OR, CONTAINS etc.) allows Domino to read them as normal words. Example: “rock and roll” finds documents containing the phrase, intact.
PARAGRAPH paragraph	This finds documents in which the words surrounding PARAGRAPH are in the same paragraph, and ranks them by how close they are. Example: “car PARAGRAPH wheels” finds documents in which “car” and “wheels” appear in the same paragraph and ranks them by how close the words are within the paragraph.
SENTENCE sentence	This finds documents in which the words surrounding SENTENCE are in the same sentence, and ranks them by how close they are. Example: “car SENTENCE wheels” finds documents in which “car” and “wheels” appear in the same sentence and ranks them by how close the words are within the sentence.

continued

<i>Operator</i>	<i>Description and examples</i>
?	This is a wildcard. It represents any single letter. It does not work with dates or numbers. Example: “?one” finds documents containing bone, cone, done, gone (and any other four-letter words that end with “one”). “???ck” finds documents containing stack, clock, stick, truck; rack, rick, rock
*	This is a wildcard. It represents any extension of letters. It does not work with dates or numbers.
TERM WEIGHT termweight	This gives importance, or “weight,” to search words. You can use any value from 0 through 65537 to assign weight.
EXACTCASE exactcase	This tells Domino to search for the exact case of the word following. Example: “exactcase Apple” finds documents containing “Apple,” but not “APPLE” or “apple.”
CONTAINS contains	This tells Domino that the field before it must contain the text after it. There should be spaces between “CONTAINS” and words surrounding it.
= < > <= >=	These help you search for numbers or dates in numeric or date fields only.
- (hyphen)	This tells Domino to find the hyphenated word pair.

The server node

The <server> node represents an installed QuickPlace server in the QuickPlace service. All actions performed on a QuickPlace server are executed from within the server node hierarchy. The server node is contained within the <service> node.

Note In this QuickPlace release you cannot perform actions on any server other than the server where the XML is executing from. If you want to perform actions on other servers in the service, you must execute the XML on each of the other servers.

Hierarchy

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
```

```
.....  
</server>  
</servers>  
</service>
```

Supported attributes

The `<server>` node supports the following attributes:

local

Syntax: `<server local="true"> </server>`

Values: "true" | "false" — Specifies whether or not the server is local to the executing XML script. In this QuickPlace release, XML must run on the local server. You must use either this attribute or the `<hostname>` element to specify the server that the XML will run on.

Supported elements

The `<server>` node supports the following named elements:

<hostname>

The `<hostname>` element is used by the server node to specify the host name that the script is executing on. The name used should be an IP address or DNS resolvable host name. The name must be the name of the local QuickPlace Server the script is being executed on.

Syntax:

```
<?xml version="1.0"?>  
<service>  
  <server>  
    <hostname>qpserver.acme.com</hostname>  
  </server>  
</service>
```

Required: Required if `<local="true">` attribute is not specified or equals "false"

Supported actions

The `<server>` node supports the following named actions:

- `getPlaceTypes`

getPlaceTypes (server)

The `getPlaceTypes` action retrieves all `PlaceTypes` that exist on the specified server. The "standard" `PlaceType` is `h_StdPlaceType`.

Syntax

```
<?xml version="1.0"?>
<service>
  <server action="getPlaceTypes">
  </server>
</service>
```

Supported attributes

The getPlaceTypes action supports the following attributes in the results:

id

Unique ID to identify the PlaceType. This value is guaranteed to be unique.

Supported elements

The getPlaceTypes action supports the following elements in the results:

<name>

Specifies the name of the PlaceType.

<description>

Provides a description of the PlaceType. This value is set in the QuickPlace UI. It is displayed during the creation of a place.

<additional_information_url>

Provides an addition information url. This value is set in the QuickPlace UI. It is displayed during the creation of a place.

Results

PlaceTypes are listed by server name. The results of the action are returned in the following format:

```
<?xml version="1.0"?>
<service>
  <server local="true">
    <placetypes>
      <placetype id="8912471890219238">
        <name>ACMETeamPlacetype</name>
        <description>The ACME Team's Placetype</description>
        <additional_information_url>
          http://www.acme.com/acmeteaminfo
        </additional_information_url>
```

```
</placetype>
<placetype>
.....
</placetype>
</placetypes>
</server>
</service>
```

The place node

The `<place>` node represents a place on a QuickPlace server in the QuickPlace service. All actions performed on a place are executed from within the `<server>` node hierarchy.

Hierarchy

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place>
          .....
        </place>
      </places>
    </server>
  </servers>
</service>
```

Supported elements

The `<place>` node supports the following named elements:

`<name>`

The `<name>` element is used by the place node to specify the name of the place being serviced. This name refers to a place on the local server executing the script.

Syntax:

```
<?xml version="1.0"?>
  <service>
    <servers>
```

```

<server local="true">
  <places>
    <place>
      <name>ACMETeam</name>
    </place>
  </places>
</server>
</servers>
</service>

```

Required: Required for all supported place actions.

<placetype>

The <placetype> element is used by the place node to specify the PlaceType that is associated with the place being serviced. The placetype element is primarily used when creating places. When performing operations involving a PlaceType, you must first identify the PlaceType within the <placetypes> node and assign it an ID. Then in the <place> node, define a <placetype> node that contains a <link> element. The link element refers to the PlaceType identified earlier.

The following example identifies an existing PlaceType and assigns it an id. Then the XML instructs that a new place be created using the PlaceType.

```

<?xml version="1.0"?>
  <service>
    <servers>
      <server local="true">
        <placetypes>
          <placetype id="ACMETeamPlacetymlink">
            <name>ACMETeamPlaceType</name>
          </placetype>
        </placetypes>
        <places>
          <place action="create">
            <name>MyPlace</name>
            <member>
              <person action="add" id="ExternalMember">

```

```

        <dn>cn=John Doe,ou=Sales,o=ACME</dn>
        </person>
    </member>
    <placetype>
        <link idref="ACMETeamPlacetypeLink">
    </placetype>
    </place>
</places>
</server>
</servers>
</service>

```

Note For more information on using PlaceType objects and creating a place, see the topics “create (place)” and “The placetype node” in this chapter.

Supported actions

The <place> node supports the following named actions:

- create
- remove
- forceRemove
- update

create (place)

The create action creates the place specified, using the PlaceType specified (optional), on the server specified. You must also specify a manager of the place, who will be the first member of the place when it is created. When you create a place, the place manager is always a person. Place creation occurs on the local server executing the script. The place must not previously exist on the server at the time of place creation or an error action status code is returned.

Syntax

```

<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <place action="create">
        <name>

```

```

    </name>
    <member>
      <person><person>
    </member>
  </place>
</server>
</servers>
</service>

```

Optional syntax

```

<?xml version="1.0"?>
  <service>
    <servers>
      <server>
        <placetypes>
          <placetype id="ACMETeamPlacetyelink">
            <name>ACMETeamPlaceType</name>
          </placetype>
        </placetypes>
        <places>
          <place action="create">
            <name></name>
            <member>
              <person></person>
            </member>
            <placetype>
              <link idref="ACMETeamPlacetyelink">
            </placetype>
          </place>
        </places>
      </server>
    </servers>
  </service>

```

Note For more information on how the <placetype> node works when creating a place, see the topic “The place node” in this chapter.

Example

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <places>
        <place action="create">
          <name>ACME_Team</name>
          <title>ACME Team Place</title>
          <members>
            <person local="true" action="add" id="LocalOwner">
              <username>JCool</username>
              <password>snoopy</password>
              <first_name>Joe</first_name>
              <last_name>Cool</last_name>
            </person>
            <person action="add" id="ExternalMember">
              <dn>cn=John Doe,ou=Sales,o=ACME</dn>
            </person>
            <group action="add" id="ExternalGroup">
              <dn>cn=Sales,ou=East,o=ACME</dn>
            </group>
          </members>
        </place>
      </places>
    </server>
  </servers>
</service>
```

remove (place)

The remove action marks the specified place for removal from the specified server. Removal of the place is performed when the `qptool remove -cleanup` command runs on the server.

Syntax

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place action="remove">
          <name>
            </name>
          </place>
        </places>
      </server>
    </servers>
  </service>
```

Example

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <places>
        <place action="remove">
          <name>AcmeTeam</name>
        </place>
      </places>
    </server>
  </servers>
</service>
```

forceRemove (place)

The forceRemove action marks the specified place for removal from the specified server and attempts to delete the files immediately. If the files are being used by another process, the files are left marked for later removal.

Syntax

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place action="forceRemove">
          <name>
            </name>
          </place>
        </places>
      </server>
    </servers>
  </service>
```

Example

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <places>
        <place action="forceRemove">
          <name>AcmeTeam</name>
        </place>
      </places>
    </server>
  </servers>
</service>
```

update (place)

The update action updates the specified information in the specified place.

Syntax

```
<?xml version="1.0"?>
<service>
  <servers>
```

```

<server>
  <places>
    <place action="update">
      <name>
        </name>
      </place>
    </places>
  </server>
</servers>
</service>

```

Supported elements

The update action supports the following elements:

<title>

Syntax: <title>*The ACME Team Place*</title>

Supported Values: Any string that represents the title of the place.

<meta_data>

Syntax:

```

<meta_data>
  <name1>value1</name1>
  <name2>value2</name2>
  <name3>value3</name3>
</meta_data>

```

Supported Values: Name/Value pairs are specified and are user-defined. The metadata Name/Value pairs are stored in the specified place as well as the Place Catalog.

Example

```

<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <places>
        <place action="update">
          <name>AcmeTeam</name>
          <title>The ACME Team Place</title>
        </place>
      </places>
    </server>
  </servers>
</service>

```

```

    <meta_data>
      <name1>value1</name1>
      <name2>value2</name2>
      <name3>value3</name3>
    </meta_data>
  </place>
</places>
</server>
</server>
</service>

```

The placetype node

The `<placetype>` node represents a PlaceType on a QuickPlace server in the QuickPlace service. In this QuickPlace release, the placetype node is primarily used when creating places. It supports no actions.

When performing operations involving a PlaceType, you must give the PlaceType an ID, then reference it in other sections of the XML. First, identify and give the PlaceType an ID, then reference the ID in other sections of the XML. For example, to create a place from the ACMETeamPlaceType, you would use the following syntax:

```

<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <placetypes>
        <placetype id="ACMETeamPlacetymlink">
          <name>ACMETeamPlaceType</name>
        </placetype>
      </placetypes>
    <places>
      <place action="create">
        <placetype>
          <link idref="ACMETeamPlacetymlink">
        </placetype>
        <name>ACME_Team</name>
      </place>
    </places>
  </server>
</servers>
</service>

```

```

<title>ACME Team Place</title>
<members>
  <person action="add" id="ExternalMember">
    <dn>cn=Charles Brown,ou=Sales,o=ACME</dn>
  </person>
</members>
</place>
</places>
</server>
</servers>
</service>

```

Supported attributes

The placetype node supports the following attributes:

id

Unique ID to identify the PlaceType.

Supported elements

The <placetype> node supports the following named elements:

<name>

The <name> element is used by the placetype node to specify the name of the placetype being serviced. This name refers to a placetype on the local server executing the script.

Syntax: <name>ACMETeamPlaceType</name>

Required: Required for all supported placetype actions.

<description>

Provides a description of the PlaceType. This value is set in the QuickPlace UI. It is displayed during the creation of a place. It is optional.

Syntax: <description>The ACME Team's Placetype</description>

<additional_information_url>

Provides an addition information url. This value is set in the QuickPlace UI. It is displayed during the creation of a place. It is optional.

Syntax: <additional_information_url>http://www.acme.com/acmeteaminfo</additional_information_url>

The person node

The <person> node represents a person on the QuickPlace server in the QuickPlace service. All actions performed on a person are executed from within the <place> node hierarchy.

Hierarchy

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place>
          <members>
            <person>
            </person>
          </members>
        </place>
      </places>
    </server>
  </servers>
</service>
```

Supported attributes

The person node supports the following attributes:

id

Syntax:

```
<person id="personid"></person>
<person idref="personid"></person>
```

Assigning a person an ID allows you to reference them in other sections of the XML. For example, if you want to create two places and add the same user as a member of both, you define and give the user an ID within the first <place> node, then reference them in the second <place> node. For example:

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
```

```

<places>
  <place action="create">
    <name>ACME_Team_Blue</name>
    <title>ACME Team Place Blue</title>
    <members>
      <person local="true" id="person1">
        <username>jdoe</username>
      </person>
      <person id="person2">
        <dn>cn=Charles Brown,ou=Sales,o=ACME</dn>
      </person>
    </members>
  </place>
  <place action="create">
    <name>ACME_Team_Red</name>
    <title>ACME Team Place Red</title>
    <rooms>
      <room>
        <name>Main.nsf</name>
        <access>
          <managers>
            <member action="add">
              <link idref="person1" />
            </member>
          </managers>
          <authors>
            <member action="remove">
              <link idref="person2" />
            </member>
          </authors>
        </access>
      </room>
    </rooms>
  </place>

```

</places>
</server>
</servers>
</service>

local

Syntax: <person local="true"></person>

Supported Values: "true" | "false" — Specifies whether or not the person is local to the specified place. A value of "true" indicates that the person exists only in the specified place. A value of "false" indicates that the person exists in a user directory, outside the specified place.

subscribed_to_newsletter

Syntax: <person subscribed_to_newsletter="true"></person>

Supported Values: "true" | "false" — Specifies whether or not the person subscribed to the place's newsletter. A value of "true" indicates that the person is subscribed. A value of "false" indicates that the person is not subscribed.

subscribed_to_calendar_events

Syntax <person subscribed_to_calendar_events="true"></person>

Supported Values: "true" | "false" — Specifies whether or not the person subscribed to the calendar events in the specified place. A value of "true" indicates that the person is subscribed. A value of "false" indicates that the person is not subscribed.

using_accessible_ui

Syntax: <person using_accessible_ui="true"></person>

Supported Values: "true" | "false" — Specifies whether or not the person is using an accessibility user interface in the specified place. A value of "true" indicates that the person is using an accessibility user interface. A value of "false" indicates that the person is not using an accessibility user interface.

email_client

Syntax: <person email_client="notes5"></person>

Supported Values: "notes5" | "outlook" | — Specifies which e-mail client the person uses. Notes5 means the person uses a Notes release 5.x mail client. Outlook means the person uses a Microsoft® Outlook® mail client.

Supported elements

The person node supports the following named elements:

<dn>

The <dn> element is used by the person node to specify the external name of the person being serviced. This name refers to a person in a directory external to QuickPlace. The format of the dn must be an LDAP distinguished name. You do not need to specify this element (nor should you) if you are operating on a person that is local to the specified place

Syntax: <dn>cn=*Jane Doe*,ou=*Sales*,o=*ACME*</dn>

Required: Required for all supported place actions if operating on an external user.

<username>

The <username> element is used by the person node to specify the person that is associated with the operation being performed. The value specified by this element represents a local user of the specified place. A local user is one that exists purely in the place and not in an external entity such as a directory. If you want to specify an external user then use the <dn> element described above.

Syntax:<username>*jdoe*</username>

Required: Person attribute local="true" must be specified.

<first_name>

The <first_name> element is used by the person node to specify the first name of the person that is associated with the operation being performed. The value specified by this element represents the first name of a local user of the specified place. This element is not applicable when the <dn> element is specified.

Syntax:<first_name>*jane*</first_name>

Required: Person attribute local="true" must be specified.

<last_name>

The <last_name> element is used by the person node to specify the last name of the person that is associated with the operation being performed. The value specified by this element represents the last name of a local user of the specified place. This element is not applicable when the <dn> element is specified.

Syntax: <last_name>*Doe*</last_name>

Required: Person attribute local="true" must be specified.

<password>

The <password> element is used by the person node to specify the password of the person that is associated with the operation being performed. The value specified by this element represents the password of a local user of the specified place. This password will be required when the specified user authenticates with the place. This element is not applicable when the <dn> element is specified.

Syntax: <password>*BigSecret*</password>

Required: Person attribute local="true" must be specified.

<phone_number>

The <phone_number> element is used by the person node to specify the phone number of the person that is associated with the operation being performed. The value specified by this element represents the phone number of a local user of the specified place. This element is not applicable when the <dn> element is specified.

Syntax: <phone_number>*978-555-1212*</phone_number>

Required: Person attribute local="true" must be specified.

<offline_password>

The <offline_password> element is used by the person node to specify the offline password of the person that is associated with the operation being performed. This password is used when the person authenticates with the place in offline mode. The value specified by this element can be used with either a local person or an external person.

Syntax: <offline_password>*BigSecret*</offline_password>

Required: N/A

<description>

Syntax: <description>*Million Dollar Sales Manager*</description>

Required: N/A

The <description> element is used by the person node to specify a description of the person that is associated with the operation being performed. The value specified by this element can be used with either a local person or an external person.

<email>

The <email> element is used by the person node to specify the e-mail address of the person that is associated with the operation being performed. The value specified by this element can be used with either a local person or an external person. This element is not applicable when the <dn> element is specified.

Syntax: <email>*jdoe@acme.com*</email>

Required: Person attribute local="true" must be specified.

<theme>

The <theme> element is used by the person node to specify the name of the theme associated with the operation being performed. The value specified by this element can be used with either a local person or an external person.

Syntax: <theme>*h_DefaultSkin*</theme>

Required: N/A

Supported actions

The person node supports the following named actions:

- add
- remove
- update

add (person)

The add action adds a person to the specified place. The person can exist in the place or can exist outside the place in an external directory, depending upon which attribute you specify for them. When adding an external person to a place, the external user directory is not consulted for existence or name correctness. You can specify any supported attributes or elements of the person when the add action is performed since the specified person is updated immediately following this add operation.

Note This action is performed to initially add a person to the specified place but it does not give that person any rights to access elements of the place. That action is performed by the <member> node within a room.

Syntax

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <places>
        <place>
          <name>
            </name>
          <members>
            <person local="true" action="add">
              <username>
```

```
        </username>
    </person>
</members>
</place>
</places>
</server>
</servers>
</service>
```

- or -

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <places>
        <place>
          <members>
            <person action="add">
              <dn></dn>
            </person>
          </members>
        </place>
      </places>
    </server>
  </servers>
</service>
```

Optional attributes

```
subscribed_to_newsletter
using_accessible_ui
subscribed_to_calendar_events
email_client
```

Optional elements

```
<password></password>
<first_name></first_name>
<last_name></last_name>
<phone_number></phone_number>
<email></email>
<offline_password></offline_password>
<theme></theme>
<description></description>
```

Example

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <places>
        <place>
          <name>ACME_Team</name>
          <members>
            <person local="true" action="add">
              <username>Jane Doe</username>
              <password>BigSecret</password>
              <first_name>Jane</first_name>
              <last_name>Doe</last_name>
            </person>
            <person action="add">
              <dn>cn=Charles Brown,ou=Sales,o=ACME</dn>
            </person>
          </members>
        </place>
      </places>
    </server>
  </servers>
</service>
```

remove (person)

The remove action removes a person from the specified place. The person can exist in the place or can exist outside the place in an external directory, depending upon which attribute you specify for them. If you remove a local person, that person is removed from the specified place. If you remove an external person, that person is removed from the place but is not removed from the external directory.

When a person is removed from a place, their membership to all rooms in the place is also removed.

Syntax

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place>
          <name>
            </name>
          </place>
        </places>
      </server>
    </servers>
  </service>
```

- and -

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place>
          <members>
            <person local="true" action="remove">
              <username>
                </username>
            </person>
```

```
    </members>
  </place>
</places>
</server>
</servers>
</service>
```

- or -

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place>
          <members>
            <person action="remove">
              <dn></dn>
            </person>
          </members>
        </place>
      </places>
    </server>
  </servers>
</service>
```

Example

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <hostname>qp.acme.com</hostname>
      <places>
        <place>
          <name>ACME_Team</name>
          <members>
```

```

    <person local="true" action="remove">
      <username>JDoe</username>
    </person>
    <person action="remove">
      <dn>cn=Charles Brown,ou=Sales,o=ACME</dn>
    </person>
  </members>
</place>
</places>
</server>
</servers>
</service>

```

update (person)

Updates a person in the specified place. When this action is called, the specified person is updated using the attributes and values you specify. You can specify any supported attributes or elements of the person when the update action is performed. No updates are performed in the external directory if the person being updated is not local.

Syntax

```

<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place>
          <name>
            </name>
          </place>
        </places>
      </server>
    </servers>
  </service>

```

- and -

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place>
          <members>
            <person local="true" action="update">
              <username>
                </username>
              </person>
            </members>
          </place>
        </places>
      </server>
    </servers>
  </service>
```

- or -

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place>
          <members>
            <person action="update">
              <dn></dn>
            </person>
          </members>
        </place>
      </places>
    </server>
```

```
</servers>
</service>
```

Optional attributes

```
subscribed_to_newsletter
using_accessible_ui
subscribed_to_calendar_events
email_client
```

Optional elements

```
<password></password>
<first_name></first_name>
<last_name></last_name>
<phone_number></phone_number>
<email></email>
<offline_password></offline_password>
<theme></theme>
<description></description>
```

Example

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <hostname>qp.acme.com</hostname>
      <places>
        <place>
          <name>ACME_Team</name>
          <members>
            <person local="true" action="update">
              <username>JDoe</username>
              <password>BiggerSecret</password>
            </person>
            <person action="update">
              <dn>cn=Charles Brown,ou=Sales,o=ACME</dn>
              <offline_password>Drats</password>
              <phone_number>978-555-1212</password>
```

```
        </person>
    </members>
</place>
</places>
</server>
</servers>
</service>
```

The group node

The `<group>` node represents a group on the QuickPlace server in the QuickPlace service. All actions performed on a group are executed from within the `<place>` node hierarchy. QuickPlace Release 3.0 does not support local groups — that is, groups that exist purely in the place. All group operations are performed on groups that have identities in external directories.

Hierarchy

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place>
          <members>
            <group>
              </group>
            </members>
          </place>
        </places>
      </server>
    </servers>
  </service>
```

Supported elements

The group node supports the following named elements:

```
<dn>
```

The <dn> element is used by the group node to specify the external name of the group being serviced. This name refers to a group in a directory external to QuickPlace. Only external groups are supported for QuickPlace Release 3.0.

Syntax: <dn>cn=*Sales*,ou=*Corporate*,o=*ACME*</dn>

Required: Required for all supported place actions.

<description>

The <description> element is used by the group node to specify a description of the group that is associated with the operation being performed. The value specified by this element must be an external group that exists in an external directory.

Syntax: <description>*The ACME Sales Team*</description>

Required: N/A

Supported actions

The <group> node supports the following named actions:

- add
- remove
- update

add (group)

The add action adds a group to the specified place. The group must exist outside the place in an external directory. When adding a group to a place, the external directory is not consulted for existence or name correctness. You can specify any supported attributes or elements of the group when the add action is performed since the specified group is updated immediately following this add operation.

Note This action is performed to initially add a group to the specified place but it does not give that group any rights to access elements of the place. That action is performed by the <member> node.

Syntax

```
<?xml version="1.0"?>
```

```
<service>
```

```
<servers>
```

```
<server>
```

```
<places>
```

```
<place>
```

```
<members>
```

```
<group action="add">
  <dn></dn>
</group>
</members>
</place>
</places>
</server>
</servers>
</service>
```

Optional attributes

The following optional attributes are supported the group node:

subscribed_to_newsletter

Specifies whether the members of the group subscribe to the place's newsletter.

Optional elements

The following optional elements are supported in the group node:

<description>

Describes the group.

Example

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <hostname>qp.acme.com</hostname>
    </server>
  </servers>
  <places>
    <place>
      <name>ACME_Team</name>
      <members>
        <group action="add">
          <dn>cn=Sales,ou=Corporate,o=ACME</dn>
        </group>
      </members>
    </place>
  </places>
```

```
</server>
</servers>
</service>
```

remove (group)

The remove action removes a group from the specified place. The group is removed from the place but is not removed from the external directory. When a group is removed from a place, their membership to all rooms in the place is also removed.

Syntax

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place>
          <members>
            <group action="remove">
              <dn></dn>
            </group>
          </members>
        </place>
      </places>
    </server>
  </servers>
</service>
```

Example

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <hostname>qp.acme.com</hostname>
      <places>
        <place>
          <name>ACME_Team</name>
```

```

    <members>
    <group action="remove">
      <dn>cn=Sales,ou=Corporate,o=ACME</dn>
    </group>
    </members>
  </place>
</places>
</server>
</servers>
</service>

```

update (group)

The update action updates a group in the specified place. When this action is called, the specified group is updated using the attributes and values you specify. You can specify any supported attributes or elements of the group when the update action is performed. No updates are performed in the external directory.

Syntax

```

<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place>
          <members>
            <group action="update">
              <dn></dn>
            </group>
          </members>
        </place>
      </places>
    </server>
  </servers>
</service>

```

Optional elements

The following optional elements are supported in the group node:

<description>

Describes the group.

Example

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <hostname>qp.acme.com</hostname>
      <places>
        <place>
          <name>ACME_Team</name>
          <members>
            <group action="update">
              <dn>cn=Sales,ou=All,o=ACME</dn>
              <description>Global Sales Team</description>
            </group>
          </members>
        </place>
      </places>
    </server>
  </servers>
</service>
```

The member node

The <member> node represents a member of one or more rooms in the specified place on the QuickPlace server in the QuickPlace service. All actions performed on a member are executed from within the <room> node hierarchy. The member node is primarily used to define, modify, or remove membership access to one or more rooms in the place. When you perform actions on a member, you must define a <person> or <group> that represents the member you are processing. Operations on a member node

are performed using an idref link to the <person> or <group> nodes previously defined in the script. An idref link relationship is demonstrated by the following:

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <places>
        <place>
          <members>
            <person local="true" id="person1">
              <username>jdoe</username>
            </person>
            <person id="person2">
              <dn>cn=Charles Brown,ou=Sales,o=ACME</dn>
            </person>
          </members>
          ...
        </places>
      <rooms>
        <room>
          <name>Main.nsf</name>
          <access>
            <managers>
              <member action="add">
                <link idref="person1"/>
              </member>
            </managers>
            <authors>
              <member action="remove">
                <link idref="person2"/>
              </member>
            </authors>
          </access>
        </room>
```

```
    </rooms>
  </place>
</places>
</server>
</servers>
</service>
```

Notice that the `<person>` node is defined first with a corresponding link ID value. That ID value is referenced through the `<link idref>` element to determine which `<person>` the `<member>` node should operate on.

Note The main distinction between a `<person>` or `<group>` and a `<member>` is that a `<person>` or `<group>` represents an entity that the place has information about. A `<member>` represents a `<person>` or `<group>` node's access or membership to a particular room.

Supported elements

The member node supports the following named elements:

`<link>`

The `<link>` element is used by the member node to provide a reference link by ID to a previously defined `<person>` or `<group>` node. The `idref` attribute is specified when `<link>` is used to reference the entity defined previously with the same value. The value specified by `idref` must match the value defined for the entity it is used to reference. For example:

```
<person id="person1"/> and <link idref="person1"/>
```

Syntax: `<link idref="person1"/>`

Required: Required for all supported `<member>` actions being performed on a `<person>` or `<group>`.

Supported actions

The member node supports the following named actions:

- add
- remove

add (member)

The add action adds a person or group with the specified access level to the specified room of the specified place. The person or group must previously exist as an entity in the place (handled by the `<person>` or `<group>` nodes) before a membership operation can be performed. When a membership action is performed, the specified entity will have immediate access to the specified room at the specified level of access.

Syntax

```
<?xml version="1.0"?>
<service>
  <servers>
    <server>
      <places>
        <place>
          <name></name>
          <members>
            <person | group id="refValue">
              </person | /group>
            </members>
            <room>
              <name></name>
              <access>
                <managers | authors | readers>
                  <member action="add">
                    <link idref="refValue"/>
                  </member>
                </managers | /authors | /readers>
              </access>
            </room>
          </place>
        </places>
      </server>
    </servers>
  </service>
```

Supported elements

Membership access level is controlled by the following elements:

<managers>

The <managers> element is used by the <room> node to specify manager access level to the room for the names specified within.

Syntax:

```
<managers>
  <member action="add">
    <link idref="refValue"/>
  </member>
</managers>
```

Required: The name of a local or external person or group that exists in the place.

<authors>

The <authors> element is used by the <room> node to specify author access level to the room for the names specified within.

Syntax:

```
<authors>
  <member action="add">
    <link idref="refValue"/>
  </member>
</authors>
```

Required: The name of a local or external person or group that exists in the place.

<readers>

The <readers> element is used by the <room> node to specify reader access level to the room for the names specified within.

Syntax:

```
<readers>
  <member action="add">
    <link idref="refValue"/>
  </member>
</readers>
```

Required: The name of a local or external person or group that exists in the place.

Example

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
```

```

<places>
  <place>
    <name>ACMETeam</name>
    <members>
      <person local="true" id="person1">
        <username>cbrown</username>
      </person>
      <person id="person2">
        <dn>cn=Jane Doe,ou=Sales,o=ACME</dn>
      </person>
    </members>
    <rooms>
      <room>
        <name>Main.nsf</name>
        <access>
          <managers>
            <member action="add">
              <link idref="person1"/>
            </member>
          </managers>
          <authors>
            <member action="remove">
              <link idref="person2"/>
            </member>
          </authors>
        </access>
      </room>
    </rooms>
  </place>
</places>
</server>
</servers>
</service>

```

remove (member)

The remove action removes a person or group access to the specified room of the specified place. The person or group must previously exist as an entity in the place (handled by the <person> or <group> nodes) before a membership operation can be performed. When a membership action is performed, the specified entity's access will be immediately removed from the specified room

Syntax

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
      <places>
        <place>
          <name></name>
          <members>
            <person | group id="refValue">
              </person | /group>
            </members>
          <room>
            <name></name>
            <member action="remove">
              <link idref="refValue" />
            </member>
          </room>
        </place>
      </places>
    </server>
  </servers>
</service>
```

Example

```
<?xml version="1.0"?>
<service>
  <servers>
    <server local="true">
```

```

<places>
  <place>
    <name>ACMETeam</name>
    <members>
      <person local="true" id="person1">
        <username>cbrown</username>
      </person>
      <person id="person2">
        <dn>cn=Jane Doe,ou=Sales,o=ACME</dn>
      </person>
    </members>

  <rooms>
    <room>
      <name>Main.nsf</name>
      <access>
        <members>
          <member action="remove">
            <link idref="person1"/>
          </member>
          <member action="remove">
            <link idref="person2"/>
          </member>
        </members>
      </access>
    </room>
  </rooms>
</place>
</places>
</server>
</servers>
</service>

```

Chapter 6

PlaceBots: Automating Tasks in a Place

This chapter describes how to automate tasks in a place using PlaceBots.

PlaceBots: automating tasks in a place

A PlaceBot is an agent, written either in Java or LotusScript®, that performs a task. PlaceBots can access, process, and manage the data in a place. For example, you can create a PlaceBot that sends e-mail to members of a place notifying them when a particular document has been edited.

You can set PlaceBots to run on a schedule, or run when a particular form is submitted. Or you can set a PlaceBot to run manually. You must have manager access to the place to create, edit, copy, delete, or run PlaceBots manually.

Because of security and performance considerations, some administrators may choose to disable the PlaceBot feature entirely. Consult your server administrator about PlaceBot policies in your organization.

You create PlaceBots using LotusScript or Java to manipulate the Domino back-end object classes. For complete documentation on the Domino object model and how to work with objects using LotusScript or Java, see the latest Domino Designer 6 Help, available on the Web at <http://www.lotus.com/ldd/doc>.

You can write, debug, and compile Java code for a PlaceBot in a Java development tool, such as Symantec Visual Cafe. You can import the .java file, or compile and import a .class or .jar file. You can also write Java or LotusScript code in any editor and import the resulting files into your place.

Creating Java PlaceBots

You can use a Java PlaceBot to access and process data in your place.

QuickPlace supports Java, Version 1.1.8 and later.

A PlaceBot written in Java may consist of one or multiple files. A Java PlaceBot file must contain a class that extends the Domino Java agent class AgentBase.

Java PlaceBot files can be of the following types:

- .java files containing Java source code. These files are compiled on the QuickPlace server when the PlaceBot is submitted through the browser.
- .class files are Java object files produced by compiling the .java files. Since these are already compiled, the files do not need to be compiled when they are submitted to the QuickPlace server. To compile your .java source agent files into .class files locally on your machine, you will need a copy of the Notes.jar files locally. Notes.jar is included with each QuickPlace server installation. If you do not have access to this file, ask your QuickPlace server administrator to make it available to you.
- .jar files are zipped, or compressed, collections of files. The .jar file generally contains one or more .class files and any other files (for example, graphic files) the PlaceBot requires.

A PlaceBot extends the AgentBase class, which extends the NotesThread class. The class that contains the PlaceBot code must be public. The entry point to the functional code must be public void NotesMain().

For more information on Java and Domino, refer to the Java information in the latest Domino Designer 6 Help, available from <http://www.lotus.com/ldd/doc>.

Example

Add the following Java code to a PlaceBot. The PlaceBot writes the name of each document it processes to the log.

```
import lotus.domino.*;
import java.util.*;
public class LogTitles extends AgentBase{
    public void NotesMain(){
        try{
            Session s = this.getSession();

            AgentContext ctx = s.getAgentContext();

            Database db = ctx.getCurrentDatabase();

            // Prepare the agent log
            Log log = s.createLog("Log");

            log.openAgentLog();
```

```

        log.setLogActions(true);
        // Get all the unprocessed documents

        DocumentCollection dc =
ctx.getUnprocessedDocuments();
        if (dc.getCount() == 0) { return; }

        // Loop thru the documents and print the
title of each document.
        Document doc = dc.getFirstDocument();

        log.logAction("Count of documents = " +
dc.getCount());
        while (doc != null)
        {
log.logAction(doc.getItemValueString("h_Name"));

                doc = dc.getNextDocument();
        }

        // Mark all the documents as processed.

        dc.updateAll();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
}

```

Creating LotusScript PlaceBots

You can use a LotusScript PlaceBot to access and process data in your place. For example, you can use LotusScript to change a value in one document based on values in other documents or to modify a place's access control list.

LotusScript is an embedded, BASIC scripting language with a powerful set of language extensions that enable object-oriented application development within and across Lotus products. LotusScript and its development tool set provide a common programming environment across Lotus applications on all platforms supported by Lotus.

LotusScript offers a wide variety of features. Its interface to Lotus products is through predefined object classes. The products oversee the compilation and loading of user scripts and automatically include class definitions to allow more efficient coding.

For more information on LotusScript and Domino, refer to the LotusScript information in the latest Domino Designer 6 Help, available from <http://www.lotus.com/ldd/doc>.

Note While QuickPlace supports LotusScript in PlaceBots, LotusScript and @formulas are not supported in hooks.

Example

Add the following LotusScript to a PlaceBot. The PlaceBot selects the current document and changes the subject line. This example would work best as a form agent.

```
Sub Initialize

    . This agent gets the document context and changes its
    subject.

    Dim session As New NotesSession

    Dim doc As NotesDocument

    Dim subj As Variant

    Dim item As NotesItem

    . Get the page being published
    Set doc = session.DocumentContext

    . Get the subject
    subj = doc.GetItemValue( "h_Name" )

    . Append a prefix string before the subject
```

```
. Note: GetItemValue always returns an array
. even if there is only a single value

Set item = doc.ReplaceItemValue( "h_Name", "Form Agent
Modified: " + subj(0))

. Save the page
Call doc.Save(True, False)

End Sub
```

Creating a PlaceBot

To create a PlaceBot, you name the PlaceBot, specify when the PlaceBot should run, and import the files for the PlaceBot. The files can be either Java or LotusScript files. Use the following steps to create a PlaceBot.

1. Click Customize.
2. In the Advanced section, choose PlaceBots.
3. Click New PlaceBot.
4. Enter a title for your PlaceBot.
5. (Optional) Enter a description of what the PlaceBot does.
6. Specify when the PlaceBot should run. For more information, see the topic “Running a PlaceBot” in this chapter.
7. Import the files for the PlaceBot.

For a LotusScript agent, import a single .lss LotusScript source file. For information about creating the file or files for a LotusScript PlaceBot, see the topic “Creating LotusScript PlaceBots” in this chapter.

For a Java agent, import one or more .java, .class or .jar files. For information on creating the file for a Java PlaceBot, see the topic “Creating Java PlaceBots” earlier in this chapter.

8. Click Done when you have filled in all the required fields.

Running a PlaceBot

When you define a PlaceBot, you must specify how the PlaceBot is activated. PlaceBots can be run when a page created from a particular form is submitted, run according to a schedule you specify, or run manually.

To run a PlaceBot when a page is submitted

Use this option to have the PlaceBot run each time a page created from a particular form is published.

1. Create the PlaceBot following the steps in the topic “Creating a PlaceBot.”
2. Select the option to run the PlaceBot when a form is submitted.
3. Choose the type of form from the drop-down list.

To run a PlaceBot at a scheduled time

Use this option to run a PlaceBot at a time you specify. This is ideal for resource-intensive PlaceBots that you want to run at off-peak times.

1. Create the PlaceBot following the steps in the topic “Creating a PlaceBot.”
2. Choose the option to run the PlaceBot on a scheduled basis.
3. Click Set Schedule.
4. Choose whether to run the PlaceBot on all pages in the current room or only on pages that are new or have been modified since the PlaceBot last ran.
5. (Optional) Choose a folder from the drop down list to limit the PlaceBot to only run on the documents in that folder.
6. Select how frequently you want the PlaceBot to run.
7. Depending on the frequency you chose, fill in the corresponding fields to specify exactly when the PlaceBot should run.
8. (Optional) Specify options that override or supplement the schedule settings. You can use these options to specify starting and stopping dates for running the PlaceBot, or you can disable the PlaceBot.
9. Click Next to submit the schedule settings, or Back to cancel your schedule settings changes and return to the previous scene.

To run a PlaceBot manually

If you have Manager access, you can run a PlaceBot manually. You might do this to test a PlaceBot that you have just created, or you may at times want to override the schedule and run the PlaceBot as needed. To run a PlaceBot manually:

1. Click Customize - PlaceBots to see the list of PlaceBots.
2. Select the PlaceBot you want to run.
3. Click Run PlaceBot.

Note You can also select a PlaceBot, click PlaceBot Log, then click Run PlaceBot from the Log file.

Using the PlaceBot log

When a PlaceBot runs, QuickPlace captures information in a log. You can include log statements in your PlaceBot code that will print information to the log. This is a helpful tool for debugging a PlaceBot or for checking that it is executing as you expect. To open the log:

1. Click Customize - PlaceBots.
2. Select the PlaceBot you want to inspect.
3. Click the PlaceBot Log button. The log displays information from the last time the PlaceBot ran and any statements you may have logged from your PlaceBot.

For an example of a PlaceBot that writes to the Log file, see the topic “Creating Java PlaceBots” earlier in this chapter.

Debugging a PlaceBot

As you define and test a PlaceBot, you may need to make some adjustments to make the PlaceBot run successfully. The following are some debugging tips for problems you might encounter.

LotusScript PlaceBots

When you run a LotusScript PlaceBot, the QuickPlace server compiles the code before executing it. Thus, the first error you might encounter would be a LotusScript compilation error or warning. The errors and warnings can be difficult to understand. For detailed information, refer to the Lotus LotusScript Release 3.1 Language Reference available as a downloadable file from <http://www.lotus.com/ldd/doc>.

If you get a compilation error, you can click Try Again to reimport the agent file after the compilation error(s) have been fixed.

Java PlaceBots

If you import Java source files (.java), you may get compilation errors when the PlaceBot is submitted to the QuickPlace server. For details on Java error messages, refer to the Java documentation.

If you get a compilation error, you can click the Back button of your browser to return to your place.

Copying a PlaceBot

You may want to copy and paste a PlaceBot from one room to another, or copy and paste within the same room. To copy a PlaceBot:

1. Click Customize - PlaceBots
2. Select the PlaceBot from the list.
3. Click Copy PlaceBot.
4. Select the room where you want the PlaceBot pasted.
5. Click Next. QuickPlace pastes a copy of the PlaceBot in the target room.

Note If you are copying a form PlaceBot to a new room, you must associate the PlaceBot with a form in the room. This is not done automatically as part of the copy and paste procedure.

Deleting a PlaceBot

Only managers can delete PlaceBots from a place. To delete a PlaceBot:

1. Click Customize - PlaceBots.
2. Select the PlaceBot you want to delete.
3. Click Delete PlaceBot.

Editing a PlaceBot

After you have tested a PlaceBot, you may want to edit it to change the name, description, files, or to modify the schedule settings. To edit a PlaceBot:

1. Click Customize - PlaceBots.
2. Select the PlaceBot you want to edit.
3. Edit the PlaceBot title, files, or schedule settings.
4. Click Done.

Disabling PlaceBots for security

PlaceBots by their nature can present a security risk. Because a PlaceBot can affect the data contained in a place, managers and server administrators must monitor them carefully. Some QuickPlace administrators may choose to disable PlaceBot functionality. Doing this disables the user interface for creating, running and editing PlaceBots, and it completely disables form PlaceBots.

Note Disabling PlaceBots does not disable existing scheduled PlaceBots because these are controlled by the Domino Agent Manager.

Running PlaceBots offline

You cannot create or edit PlaceBots in the offline environment. You also cannot run a PlaceBot manually offline. Scheduled PlaceBots do not run in the offline environment.

Form PlaceBots continue to run as usual.

Chapter 7

Customizing the Look of a Place

This chapter describes how to use themes and HTML to make a place look the way you want.

Customizing the look of a place

When you create a place, you can select its look and the layout by choosing from a gallery of predefined “themes.” The theme controls the look of a place by determining aspects such as fonts and background colors, how an element looks when it is selected, and where the navigational controls appear.

You can create new themes based on existing themes or from scratch. By creating a custom theme, you can give your place a strong brand identity, design it to look like other corporate sites, supply additional functionality, or just give it a unique look.

Themes are implemented using the QuickPlace skins architecture and are defined using HTML. So to customize an existing theme, you just edit the HTML and upload the modified files. QuickPlace provides a set of custom HTML tags that you can use to define the elements in each layout.

You can create a custom theme from an existing theme with minimal HTML development skills. To create a theme from scratch, however, requires more advanced expertise with HTML and Web site development.

When you customize a theme, you can take advantage of all of the power of HTML to add functionality to a place. Here are some ways you might enhance a place using themes:

- Apply the corporate brand identity to a place or create a custom graphic identity for a collaborative application.
- Integrate a place seamlessly as a collaborative component within a larger corporate Web site.
- Provide links from a place to other Web sites such as corporate Web sites, eCommerce sites, or to customer support services.

- Make new features available by embedding ActiveX® controls or Java applets in the custom theme.
- Use JavaScript to program dynamic effects into the custom theme.

Theme layouts

Each theme is composed of a group of layouts that define the appearance of specific place components. For example, the layout for a page differs from the layout of a folder. But they will probably share some style elements as part of a common theme. A theme is composed of the following layouts and stylesheet:

<i>Layout</i>	<i>File type</i>	<i>Purpose</i>
Page	.htm	Defines the appearance of a page being read
Page editing	.htm	Defines the appearance of a page being edited
List folder	.htm	Defines the appearance of a List or Response folder
Headlines folder	.htm	Defines the appearance of a Headlines folder
Slideshow folder	.htm	Defines the appearance of a Slideshow folder
Stylesheet	.css	Defines styles such as fonts and colors for all layouts

Note In most cases, you can use a single theme to customize the look of a page, list folder, and slideshow folder.

Customizable components

The following table shows the components you can customize for each layout.

<i>Component Name</i>	<i>Page</i>	<i>List folder</i>	<i>Slideshow folder</i>	<i>Headlines folder</i>	<i>Edit</i>
Logo	x	x	x	x	x
Page content	x	x	x	x	x
Actions	x	x	x	x	x
Help	x	x	x	x	x
TOC	x	x	x	x	
Path	x	x	x	x	
QuickSearch	x	x	x	x	
WhatsNew	x	x	x	x	
AdvancedSearch	x	x	x	x	
SignIn	x	x	x	x	
Offline	x	x	x	x	

continued

<i>Component Name</i>	<i>Page</i>	<i>List folder</i>	<i>Slideshow folder</i>	<i>Headlines folder</i>	<i>Edit</i>
Chat	x	x	x	x	x
Notify	x	x	x	x	
Print	x	x	x	x	
PageTitle	x	x	x	Note 1	x
Navigation	x	x	x	Note 2	
Jump	Note 3	x	x	Note 2	
AuthorAndModified	x	Note 3	x	x	
Revision	x	Note 3	x	x	
HeadlinesFolder				x	
MyStatus	x	x	x	x	x
SiteMapLauncher	x	x	x	x	
DownloadFile	x		x	x	
MyPlaces	x	x	x	x	

Notes

- You can also import a JPEG or GIF graphic file to represent a theme in the Custom Theme Gallery.
- Although you can optionally include the PageTitle component in a Headlines folder, you would normally omit this component and display the page title prominently instead.
- Do not use the Navigation and Jump components in the Headlines Folder layout because the Headlines Folder is designed to provide a headlines style of navigation in place of the previous/next navigation used in other folder types.
- You can include the Jump component in the Page layout and you can include the AuthorAndModified and Revision components in the ListFolder layout. These components will all display as “empty,” using the HTML parameter emptyFormat.

Creating a custom theme from an existing theme

The simplest way to customize a theme is to find a theme that’s close to what you want. Then extract the theme’s HTML source files, customize them, and upload the modified files as a custom theme. To do this:

1. Open a place.
2. Click Customize - Decorate - “Choose a theme.”
3. Select the theme that most closely represents the look and layout you want for your place and click Next.

4. QuickPlace applies the theme to your place.
5. Select Customize - Custom Themes - New Theme.
6. Enter a title for the theme you are creating and click Next.
7. QuickPlace returns you to the Custom Themes page.
8. Select your theme, then click it.
9. The Edit Theme page displays the file associated with each layout.

To view or modify the source code for a layout, drag the file to your desktop and open it in an HTML editor. If you are using an editor such as HomeSite that supports in-place editing, you can right-click a file name and choose the editor from the right-click menu. This opens the editor within QuickPlace. Changes you make to the HTML file are automatically uploaded when you save and exit the editor.

You can also edit the original source file in an HTML editor, and click the Reload button from the Edit Theme page to reload the modified file.

Creating a theme from scratch

To create a theme from scratch, do the following:

1. Choose Customize - Custom Themes - New Theme.
2. Enter a title for the theme.
3. (Optional) Enter a description for the theme.
4. Click the Browse button to locate .css or .htm files for the layout.
5. Select the file from the file system and click OK to upload the .htm file.
6. Click Next to save the theme.

To generate layout files

As you develop a theme, QuickPlace can take the code from one layout and apply it to all layouts for which you have not explicitly supplied a file. This is a shortcut for applying a common look and feel to multiple layouts.

For example, suppose you are creating a place called Haiku. You might start by creating the look and layout you want for a page being read. You can then use this file to generate files for the other layouts. You might have only minor modifications to make to the layout for a page being edited, with possibly more extensive modifications for the various folder styles.

This feature also lets you develop a custom theme in stages, replacing generated layouts with custom files as the theme progresses.

Do the following to create a theme by generating layout files.

1. In the New Theme page, click Browse to locate the HTML file for an existing layout.
2. Select Generate under the other layouts. Layout files are generated based on the existing file.
3. Edit the generated files.
4. Click Reload in your browser to update the layout files.
5. Click Next.

To edit a custom theme

The option to edit a custom theme is not available until you have created one or more custom themes.

1. Choose Customize - Custom Themes.
2. Select your theme, then click it.
3. In the Edit theme page, you can modify the name or description, edit the files that make up the custom theme, or replace files.
4. Click Next to save your changes, or Back to return to the Custom Themes page.

To delete a custom theme

Deleting a custom theme removes all of the associated files from the place.

1. Choose Customize - Custom Themes.
2. Select the custom theme.
3. Click Delete Theme.

Note You can also delete a custom theme by clicking Delete Theme button from the Edit Theme page.

To reuse a custom theme

If you customize a theme, you may want to keep the theme as part of a template from which you can build similar places. To do this, save the place containing the custom theme as a custom PlaceType, which you can then use for creating new QuickPlace applications.

For information on creating a PlaceType, see the “PlaceTypes: Using a Place as a Design Template” chapter.

Note You can only create custom themes using Internet Explorer.

Creating a layout using the <skincomponent> tag

The HTML tag that controls the style and placement of elements in a place layout is the <SkinComponent tag>. The basic syntax for the <SkinComponent tag> tag is as follows:

```
name="<skincomponentname>" (required)
format="<format html>" (optional)
selectedformat="<format html>" (optional)
emptyformat="<html>" (optional)
delimiter="<html>" (optional)
PrefixHTML="<html>" (optional)
PostfixHTML="<html>" (optional)
ReplaceString="<STRING_1=REPLACEMENT_1 && ... && ...>"
(optional) >
```

<i>Attribute</i>	<i>Description</i>
name	Required. Specifies the name of the theme component you are modifying. Valid names are described below.
format	The format HTML. The keyword is replaced for each relevant entry
selectedformat	Same as format but it applies to the selected value. For example, the format of the selected TOC entry or the selected headlines folder entry.
emptyformat	What is returned when there are no values to iterate over.
delimiter	The HTML placed between each of the items in a list of values.
PrefixHTML	The HTML placed before each of the values in a list.
PostfixHTML	The HTML placed after each of the values in a list.
Replacestring	Finds and replaces one or more strings with replacement strings.

Component name

The name attribute can be one of the following:

- SceneActions — the actions associated with the current page
- RoomActions — the actions associated with the current room
- FolderActions — the actions associated with the current folder
- Chat
- Help
- Logo
- Notify

- Search
- SignIn
- Path
- TOC
- Navigation
- Jump
- PageTitle
- WhatsNew
- Revision
- HeadlinesFolder
- PageContent — only supports the name attribute

Usage

The tag allows you to identify a piece of the QuickPlace user interface and modify the look and placement of that element. By modifying various elements and adding HTML or JavaScript within the tag, you can significantly customize the look and functionality of a QuickPlace application.

The attributes PrefixHTML, PostfixHTML, emptyformat, and delimiter work together to help you control what displays in a particular context. For example, you may want an HTML string to offer a set of instructions that go with a set of action buttons. When the action buttons are hidden, the text should be hidden as well.

Creating a layout using the <IteratingValue> tag

Many of the components will contain a list of values, such as the items in a Table of Contents. In these cases, you can use the HTML tag within the tag to iterate through the values in a list. The basic syntax for the <IteratingValue tag> tag is as follows:

```
<attribute="anchor | anchor.href | anchor.text | anchor.selected"
(optional)
class="class name" (optional)>
```

<i>Attribute</i>	<i>Description</i>
attribute	All or part of a fully qualified HTML link for the iterating value in a list. See the list of anchor types below.
class	The name of the class defined in an associated style sheet. The class name is inserted into the anchor information for the iterating value.

Attribute types

The attribute describing the HTML link can take one of the following forms:

- `anchor` returns all of the HTML that describes the iterating value, including the URL, and associated text. For example, `anchor.href` returns the URL for the value. For example, “`www.lotus.com.`”
- `anchor.text` returns text associated with the value, for example “`lotus.`”
- `anchor.selected` returns true if the value is selected, false if it is not.

Usage

Use the tag to select a value in a list. The attribute for the value identifies all or part of the HTML link that describes a particular value in a list. Use the class attribute to add styles defined as a class in an associated style sheet.

Using the same HTML in multiple layouts

Because the Page, ListFolder, and Slideshow layouts share so many common components, you can create one HTML file that applies styles to these three layouts. You create the HTML for the Slideshow Folder, which contains the superset of components used in the three layouts. To control how the non-applicable components display for a layout — for example, the Jump component for the Page layout, and the AuthorAndModified and Revision components for the ListFolder — you can achieve various results by setting the `emptyFormat`, `prefixHTML`, and `postfixHTML` parameters.

For example, if you want the empty components to occupy the same vertical space as they do when in use, set the parameter as follows:

```
emptyFormat = “&nbsp;”
```

If you place each component in a separate table row, you can have the component’s row “collapse” when it is empty, so that it occupies no space. Given that the `prefixHTML` and `postfixHTML` parameters are not output when the component is empty, you can use these parameters to provide the following table structure, as follows:

```
emptyFormat = “”  
prefixHTML = “”  
postfixHTML = “”
```

Chapter 8

PlaceTypes: Using a Place as a Design Template

This chapter describes how to save the design of a place as a PlaceType and use it to create new places.

PlaceTypes: using a place as a template

As you set up a place to meet the needs of your team or organization, you may want to preserve your customizations for use in other places. For example, if you create a theme that gives a particular place the look and feel of your corporate Web site, you may want to make that design available for creating other places in your organization.

You can preserve the design and structure of a place by creating a custom PlaceType. A PlaceType is a blueprint from which users can create new places. If you are familiar with Domino/Notes architecture, a PlaceType is a Domino database template.

Creating a PlaceType and making it available to users is a two-step process. A user with Manager access to a place customizes the place, allows it to be a PlaceType, and specifies what design elements will be preserved in the PlaceType. Then a server administrator must actually create the PlaceType on the server so it is presented as a choice to users. For more information on creating and administering PlaceTypes, see the *QuickPlace Administrator's Guide*.

To allow a place to be a PlaceType

You must have Manager access to designate a place as a PlaceType. To create a PlaceType:

1. Open the place.
2. Click Customize.
3. Click PlaceType Options in the Advanced Customization Features section.
4. Click Edit.

5. Select Yes to allow PlaceTypes to be created from the place. The default selection of No prevents the administrator from listing the place as a PlaceType on the server.
6. (Optional) Enter a text description for the PlaceType.
7. (Optional) Click Browse to import a GIF or JPEG image file, no larger than 100 pixels by 80 pixels, that contains a “thumbnail sketch” of a page in the place.
8. (Optional) Enter a URL that links users to a Web page providing details about the PlaceType and how to use it. If you specify the address of a Web page, the linked text “More info” is displayed below the description of the PlaceType in the list of PlaceTypes.

If the Web page is on the current QuickPlace server, you can enter an abbreviated address that begins with a / (forward slash). For example, if the current QuickPlace server is called TestServer and you enter /quickplace/acme/main.nsf, the address will be interpreted as http://testserver/quickplace/acme/main.nsf.

If the Web page is on another server, enter the full address of the Web page. For example, if the page is in a place called Acme on a server called HighTestServer, enter http://hightestserver/quickplace/acme/main.nsf.

9. Choose Yes to include the current list of members as part of the PlaceType, or No to save the PlaceType with no members.
10. Choose Yes to allow managers of places created from the PlaceType to make changes to the table of contents.
11. Uncheck any features that you do not want included in the PlaceType. For example, unchecking the PlaceBot option means that PlaceBots will be unavailable in places created from this PlaceType.
12. Click Next.
13. Tell the administrator to add the PlaceType to the server.

Index

A

Architecture, 3-1
Automating tasks, 6-1

D

Databases, 3-2
Debugging
 PlaceBots, 6-7
Directory structure, 3-2

H

HTML
 customizing themes with, 7-1
 using in multiple layouts, 7-8
 viewing a layout's source
 code, 7-4
HTML tags
 <IteratingValue> tag, 7-7
 <skincomponent> tag, 7-6

J

Java
 creating PlaceBots with, 6-1

L

Layouts
 definition and table of, 7-2
 examples of, 7-2
Log file
 using the PlaceBot log, 6-7
LotusScript
 creating PlaceBots with, 6-1, 6-4

M

Members directory
 creating, 3-3

O

Objects
 comparison with Domino
 objects, 3-1
Offline
 running PlaceBots, 6-9

P

PlaceBots
 copying, 6-8
 creating, 6-5
 creating with Java, 6-1
 creating with LotusScript, 6-1, 6-4
 debugging, 6-7
 deleting, 6-8
 disabling for security, 6-8
 editing, 6-8
 overview, 6-1
 running, 6-5
 running offline, 6-9
 the PlaceBot log, 6-7
PlaceTypes
 definition, 8-1

S

Security
 disabling PlaceBots, 6-8

T

Templates, 3-2
 administration, 3-3
Themes
 creating new from existing, 7-3
 deleting, 7-5
 HTML and, 7-1
 overview, 7-1

