MobiLink
Server-Initiated Synchronization

June 2008
Version 11.0.0
Copyright and trademarks

Copyright © 2008 iAnywhere Solutions, Inc. Portions copyright © 2008 Sybase, Inc. All rights reserved.

This documentation is provided AS IS, without warranty or liability of any kind (unless provided by a separate written agreement between you and iAnywhere).

You may use, print, reproduce, and distribute this documentation (in whole or in part) subject to the following conditions: 1) you must retain this and all other proprietary notices, on all copies of the documentation or portions thereof, 2) you may not modify the documentation, 3) you may not do anything to indicate that you or anyone other than iAnywhere is the author or source of the documentation.

iAnywhere®, Sybase®, and the marks listed at http://www.sybase.com/detail?id=1011207 are trademarks of Sybase, Inc. or its subsidiaries. ® indicates registration in the United States of America.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.
## Contents

About this book ................................................................................................. v

About the SQL Anywhere documentation .......................................................... vi

1. Introducing server-initiated synchronization ........................................ 1
   Introduction to server-initiated synchronization ................................................ 2
   Components of server-initiated synchronization .............................................. 4
   Supported platforms for server-initiated synchronization ............................... 5
   Deployment of server-initiated synchronization .............................................. 6
   Quick start to server-initiated synchronization ............................................... 7

2. Setting up server-initiated synchronization ........................................... 9
   Push requests ..................................................................................................... 10
   Setting properties ............................................................................................ 14
   Notifiers ........................................................................................................... 17
   Gateways and carriers ..................................................................................... 20
   Device tracking ............................................................................................... 22
   Listeners ........................................................................................................... 28

3. Listener utility ............................................................................................... 37
   Listener syntax ............................................................................................... 38

4. Listeners for Palm devices ........................................................................ 49
   Palm Listener utilities ...................................................................................... 50

5. MobiLink Notifier properties ..................................................................... 53
   Common properties ......................................................................................... 54
   Notifier properties .......................................................................................... 55
   Gateway properties .......................................................................................... 66
   Carrier properties ........................................................................................... 74
6. Server-initiated synchronization system procedures ......................... 77

Introduction to server-initiated synchronization system procedures .......... 78
IBM DB2 mainframe server-initiated synchronization system procedure name conversions ............................................................................................................... 79
ml_delete_device .......................................................................................... 80
ml_delete_device_address ......................................................................... 81
ml_delete_listening ..................................................................................... 82
ml_set_device .............................................................................................. 83
ml_set_device_address .............................................................................. 85
ml_set_listening .......................................................................................... 87

7. MobiLink Listener SDK for Palm ........................................................... 89

Introduction to MobiLink Listener SDK for Palm ........................................ 90
Message processing interface .......................................................................... 91
Device dependent functions ........................................................................... 100

I. Glossary .................................................................................................. 107

8. Glossary ................................................................................................... 109

Index ......................................................................................................... 139
About this book

Subject
This book describes MobiLink server-initiated synchronization, a feature of MobiLink that allows you to initiate synchronization or other remote actions from the consolidated database.

Audience
This book is for MobiLink users who want to use this advanced feature.

Before you begin
For more information about MobiLink, see MobiLink - Getting Started [MobiLink - Getting Started].
About the SQL Anywhere documentation

The complete SQL Anywhere documentation is available in three formats that contain identical information.

- **HTML Help**  The online Help contains the complete SQL Anywhere documentation, including the books and the context-sensitive help for SQL Anywhere tools. The online Help is updated with each release of the product.

  If you are using a Microsoft Windows operating system, the online Help is provided in HTML Help (CHM) format. To access the documentation, choose Start » Programs » SQL Anywhere 11 » Documentation » Online Books.

  The administration tools use the same online documentation for their Help features.

- **Eclipse**  On Unix platforms, the complete online Help is provided in Eclipse format. To access the documentation, run sadoc from the bin32 or bin64 directory of your SQL Anywhere 11 installation.

- **PDF**  The complete set of SQL Anywhere books is provided as a set of Portable Document Format (PDF) files. You must have a PDF reader to view information. To download Adobe Reader, visit Adobe.com.

  To access the PDF documentation on Microsoft Windows operating systems, choose Start » Programs » SQL Anywhere 11 » Documentation » Online Books - PDF Format.

  To access the PDF documentation on Unix operating systems, use a web browser to open install-dir/documentation/en/pdf/index.html.

About the books in the documentation set

**SQL Anywhere documentation**

The SQL Anywhere documentation consists of the following books:

- **SQL Anywhere 11 - Introduction**  This book introduces SQL Anywhere 11—a comprehensive package that provides data management and data exchange, enabling the rapid development of database-powered applications for server, desktop, mobile, and remote office environments.

- **SQL Anywhere 11 - Changes and Upgrading**  This book describes new features in SQL Anywhere 11 and in previous versions of the software.

- **SQL Anywhere Server - Database Administration**  This book describes how to run, manage, and configure SQL Anywhere databases. It describes database connections, the database server, database files, backup procedures, security, high availability, and replication with Replication Server, as well as administration utilities and options.

- **SQL Anywhere Server - Programming**  This book describes how to build and deploy database applications using the C, C++, Java, PHP, Perl, Python, and .NET programming languages such as Visual Basic and Visual C#. A variety of programming interfaces such as ADO.NET and ODBC are described.

- **SQL Anywhere Server - SQL Reference**  This book provides reference information for system procedures, and the catalog (system tables and views). It also provides an explanation of the SQL Anywhere implementation of the SQL language (search conditions, syntax, data types, and functions).
• **SQL Anywhere Server - SQL Usage**  This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.

• **MobiLink - Getting Started**  This book introduces MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.

• **MobiLink - Client Administration**  This book describes how to set up, configure, and synchronize MobiLink clients. MobiLink clients can be SQL Anywhere or UltraLite databases. This book also describes the Dbmlsync API, which allows you to integrate synchronization seamlessly into your C++ or .NET client applications.

• **MobiLink - Server Administration**  This book describes how to set up and administer MobiLink applications.

• **MobiLink - Server-Initiated Synchronization**  This book describes MobiLink server-initiated synchronization, a feature of MobiLink that allows you to initiate synchronization or other remote actions from the consolidated database.

• **QAnywhere**  This book describes QAnywhere, which is a messaging platform for mobile and wireless clients, as well as traditional desktop and laptop clients.

• **SQL Remote**  This book describes the SQL Remote data replication system for mobile computing, which enables sharing of data between a SQL Anywhere consolidated database and many SQL Anywhere remote databases using an indirect link such as email or file transfer.

• **UltraLite - Database Management and Reference**  This book introduces the UltraLite database system for small devices.

• **UltraLite - C and C++ Programming**  This book describes UltraLite C and C++ programming interfaces. With UltraLite, you can develop and deploy database applications to handheld, mobile, or embedded devices.

• **UltraLite - M-Business Anywhere Programming**  This book describes UltraLite for M-Business Anywhere. With UltraLite for M-Business Anywhere you can develop and deploy web-based database applications to handheld, mobile, or embedded devices, running Palm OS, Windows Mobile, or Windows.

• **UltraLite - .NET Programming**  This book describes UltraLite.NET. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.

• **UltraLiteJ**  This book describes UltraLiteJ. With UltraLiteJ, you can develop and deploy database applications in environments that support Java. UltraLiteJ supports BlackBerry smartphones and Java SE environments. UltraLiteJ is based on the iAnywhere UltraLite database product.

• **Error Messages**  This book provides a complete listing of SQL Anywhere error messages together with diagnostic information.

### Documentation conventions

This section lists the typographic and graphical conventions used in this documentation.
Syntax conventions

The following conventions are used in the SQL syntax descriptions:

- **Keywords**  All SQL keywords appear in uppercase, like the words ALTER TABLE in the following example:

  ```sql
  ALTER TABLE [ owner.]table-name
  ```

- **Placeholders**  Items that must be replaced with appropriate identifiers or expressions are shown like the words `owner` and `table-name` in the following example:

  ```sql
  ALTER TABLE [ owner.]table-name
  ```

- **Repeating items**  Lists of repeating items are shown with an element of the list followed by an ellipsis (three dots), like `column-constraint` in the following example:

  ```sql
  ADD column-definition [ column-constraint, ... ]
  ```

  One or more list elements are allowed. In this example, if more than one is specified, they must be separated by commas.

- **Optional portions**  Optional portions of a statement are enclosed by square brackets.

  ```sql
  RELEASE SAVEPOINT [ savepoint-name ]
  ```

  These square brackets indicate that the `savepoint-name` is optional. The square brackets should not be typed.

- **Options**  When none or only one of a list of items can be chosen, vertical bars separate the items and the list is enclosed in square brackets.

  ```sql
  [ ASC | DESC ]
  ```

  For example, you can choose one of ASC, DESC, or neither. The square brackets should not be typed.

- **Alternatives**  When precisely one of the options must be chosen, the alternatives are enclosed in curly braces and a bar is used to separate the options.

  ```sql
  [ QUOTES { ON | OFF } ]
  ```

  If the QUOTES option is used, one of ON or OFF must be provided. The brackets and braces should not be typed.

Operating system conventions

SQL Anywhere runs on a variety of platforms, including Windows, Windows Mobile, Unix, Linux, and Mac OS X. To simplify references to operating systems, the documentation groups the supported operating systems as follows:

- **Windows**  The Microsoft Windows family of operating systems for desktop and laptop computers. The Windows family includes Windows Vista and Windows XP.

- **Windows Mobile**  Windows Mobile provides a Windows user interface and additional functionality, such as small versions of applications like Word and Excel. Windows Mobile is most commonly used on mobile devices.
Limitations or variations in SQL Anywhere are commonly based on the underlying operating system, and seldom on the particular variant used (Windows Mobile).

- **Unix** Unless specified, Unix refers to Linux, Mac OS X, and Unix platforms.

## File name conventions

In many cases, references to file and directory names are similar on all supported platforms, with simple transformations between the various forms. To simplify the documentation in these cases, Windows conventions are used. In other cases, where the details are more complex, the documentation shows all relevant forms.

Here are the conventions used to simplify the documentation of file and directory names:

- **install-dir** During the installation process, you choose where to install SQL Anywhere. The documentation refers to this location using the convention `install-dir`.

  After installation is complete, the environment variable SQLANY11 specifies the location of the installation directory containing the SQL Anywhere components (`install-dir`).

  For example, the documentation may refer to a file as `install-dir\readme.txt`. On Windows platforms, this reference is equivalent to `%SQLANY11%\readme.txt`. On Unix platforms, this reference is equivalent to `$SQLANY11/readme.txt`.

  For more information about the default location of `install-dir`, see “SQLANY11 environment variable” [SQL Anywhere Server - Database Administration].

- **Uppercase and lowercase directory names** On Windows, directory names often use mixed case. References to directory names can use any case, since the Windows file system is not case sensitive.

  Unix file systems are case sensitive. The use of mixed-case is less common.

  The SQL Anywhere installation program follows operating system conventions for its directory structure. On Windows, the installation contains directories such as Bin32 and Documentation. On Unix, these directories are called bin32 and documentation.

  The documentation often uses the mixed case forms of directory names. Usually, you can convert a mixed case directory name to lowercase for the equivalent directory name on Unix platforms. For example, the directory MobiLink is mobilink on Unix platforms.

- **Slashes separating parts of directory names** The documentation uses backslashes as the directory separator. For example, the PDF form of the documentation is found in the directory `install-dir\Documentation\en\pdf`, which is the Windows form.

  On Unix platforms, replace the backslash with the forward slash. The PDF documentation is found in the directory `install-dir/Documentation/en/pdf`.

- **Executable files** The documentation shows executable file names using Windows conventions, with a suffix such as `.exe` or `.bat`. On Unix platforms, executable file names have no suffix.

  For example, on Windows, the network database server is `dbsrv11.exe`. On Unix, Linux, and Mac OS X, it is `dbsrv11`.
samples-dir  The installation process allows you to choose where to install the samples that are included with SQL Anywhere, and the documentation refers to this location using the convention `samples-dir`.

After installation is complete, the environment variable SQLANYSAMP11 specifies the location of the directory containing the samples (`samples-dir`). From the Windows Start menu, choosing Programs » SQL Anywhere 11 » Sample Applications And Projects opens a Windows Explorer window in this directory.

For more information about the default location of `samples-dir`, by operating system, see “Samples directory” [SQL Anywhere Server - Database Administration].

**Environment variables**  The documentation refers to setting environment variables. On Windows, environment variables are referred to using the syntax `%ENVVAR%`. On Unix, Linux, and Mac OS X, environment variables are referred to using the syntax `$ENVVAR` or `$ENVVAR`.

Unix, Linux, and Mac OS X environment variables are stored in shell and login startup files, such as `.cshrc` or `.tcshrc`.

---

**Graphic icons**

The following icons are used in this documentation.

- A client application.

- A database server, such as Sybase SQL Anywhere.

- A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.

- Replication or synchronization middleware. These assist in sharing data among databases. Examples are the MobiLink server and the SQL Remote Message Agent.
Contacting the documentation team

We would like to receive your opinions, suggestions, and feedback on this document.

To submit your comments and suggestions, send an email to the SQL Anywhere documentation team at iasdoc@sybase.com. Although we do not reply to emails, your feedback helps us to improve our documentation, so your input is welcome.

Finding out more and requesting technical support

Additional information and resources, including a code exchange, are available at the Sybase iAnywhere Developer Community at http://www.sybase.com/developer/library/sql-anywhere-techcorner.

If you have questions or need help, you can post messages to the Sybase iAnywhere newsgroups listed below.

When you write to one of these newsgroups, always provide detailed information about your problem, including the build number of your version of SQL Anywhere. You can find this information by running the following command: `dbeng11 -v`.

The newsgroups are located on the forums.sybase.com news server. The newsgroups include the following:

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianynwhere.public.sqlanywhere.qanywhere
Newsgroup disclaimer

iAnywhere Solutions has no obligation to provide solutions, information, or ideas on its newsgroups, nor is
iAnywhere Solutions obliged to provide anything other than a systems operator to monitor the service and
ensure its operation and availability.

iAnywhere Technical Advisors, as well as other staff, assist on the newsgroup service when they have time
available. They offer their help on a volunteer basis and may not be available on a regular basis to provide
solutions and information. Their ability to help is based on their workload.
CHAPTER 1

Introducing server-initiated synchronization

Contents

Introduction to server-initiated synchronization ...................................................... 2
Components of server-initiated synchronization ..................................................... 4
Supported platforms for server-initiated synchronization ...................................... 5
Deployment of server-initiated synchronization ...................................................... 6
Quick start to server-initiated synchronization ...................................................... 7
Introduction to server-initiated synchronization

Server-initiated synchronization allows you to initiate MobiLink synchronization from the consolidated database. This means you can push data updates to remote databases, as well as cause remote databases to upload data to the consolidated database. This MobiLink component provides programmable options for determining what changes in the consolidated database initiate synchronization, how remotes are chosen to receive push messages, and how the remotes respond.

Example

For example, a fleet of truck drivers uses mobile databases to determine routes and delivery points. A driver synchronizes a report of a traffic disruption. A component called the Notifier detects the change in the consolidated database and automatically sends a message to the remote device of every driver whose route is affected, which causes the drivers' remote databases to synchronize so that the drivers use an alternate route.

The notification process

In the following illustration, the Notifier polls a consolidated database and detects a change that it has been configured to look for. In this scenario, the Notifier sends a message to a single remote device, resulting in the remote database being updated via synchronization.

Following are the steps that occur in this example:

1. Using a query based on business logic, the Notifier polls the consolidated database to detect any change that needs to be synchronized to the remote.
2. When a change is detected, the Notifier prepares a message to send to the remote device.
3. The Notifier sends the message. By default, it uses the same protocol as you use for synchronization. Alternatively, you can set up UDP or SMTP gateways for it to use.

4. The Listener checks the subject, content, and sender of the message against a filter.

5. If the message matches the filter, the Listener runs a program that has been associated with the filter. For example, the Listener runs dbmlsync or it launches an UltraLite application.

**Connection-initiated synchronization**

In addition to initiating synchronization on the server, you can also initiate synchronization using internal messages that are generated by the Listener on the remote device. These internal messages indicate a change in connectivity, such as when a device enters Wi-Fi coverage, the user makes a RAS connection, or the user puts the device in the cradle.

See “Connection-initiated synchronization” on page 32.
Components of server-initiated synchronization

MobiLink server-initiated synchronization uses the following components:

- **Push requests** cause synchronization to occur. A push request takes the form of some data that you insert into a table on the MobiLink consolidated database, or in some cases data inserted into a temporary table or even just a SQL result set. You can create push requests in any way that you cause data to be inserted into a table. For example, a push request could be created by a database trigger that is activated when a price changes. Any database application can create push requests, including the Notifier. See “Push requests” on page 10.

- **The Notifier** is a program running on the same computer as the MobiLink server. It polls the consolidated database on a regular basis, looking for push requests. You control how often the Notifier polls the database. You specify business logic that the Notifier uses to gather push requests, including which remote devices should be notified. When the Notifier detects a request, it sends the message associated with the request to a Listener on one or more remote devices. You have the option to send repeatable messages with an expiry time. See “Notifiers” on page 17.

- **The Listener** is a program that is installed on each remote device. It receives messages from the Notifier and initiates action. The action is usually synchronization, but can be other things. You can configure the Listener to act only on messages from selected sources, or with specific content.

  On Windows or Windows Mobile, the Listener is an executable program configured by command line options. To receive a message, the remote device must be on and the Listener must be started. See “Listener utility” on page 37.

  On the Palm OS, you first create a configuration file by running the Palm Listener Configuration utility on a Windows desktop. You then copy the configuration file to your Palm device and run the Palm Listener. See “Listeners for Palm devices” on page 49.

- **Gateways** provide an interface to send messages from the Notifier to the Listener. You can send messages using a SYNC gateway, a UDP gateway, or an SMTP gateway. The SYNC gateway uses the same protocol as your MobiLink synchronizations.

  - **Device tracking gateways** provide a way to automatically track remote devices. Using device tracking functionality, you don't have to know the addresses of remote devices. You supply the gateway name of your device tracker gateway (by default, Default-DeviceTracker) and the MobiLink user name, and MobiLink routes the message through the appropriate gateway to the appropriate device.

For more information, see “Gateways and carriers” on page 20.
Supported platforms for server-initiated synchronization

See:

- SQL Anywhere Supported Platforms and Engineering Support Status
Deployment of server-initiated synchronization

Following are some issues that you should consider before deploying server-initiated synchronization applications.

**Limitations of Listeners when using UDP gateways**
- On UDP gateways, the Listener keeps a socket open for listening, and so must be connected to an IP network to receive notifications.
- The IP address on the remote device needs to be reachable from the MobiLink server.

**Limitations of Listeners on Windows, including Windows Mobile**
- The current set of supported wireless modems require that the operating system is running, which could result in battery drain. Make sure that you have enough power for your usage pattern.

**Palm Listeners can't automatically use device tracking**
- On the Palm, device tracking does not work automatically. However, there is a way to enable it.
  See “Using device tracking with Listeners that don't support it” on page 24.
Quick start to server-initiated synchronization

The following steps provide an overview of the tasks required to set up server-initiated synchronization, assuming that you already have MobiLink synchronization set up.

Overview of setting up server-initiated synchronization

1. Create a table to store push requests on the consolidated database.
   See “Push requests” on page 10.
2. Set up the Notifier to create and manage push requests.
   See “Notifiers” on page 17.
3. Set up the Listener to filter and act on Notifier push requests.
   See “Listeners” on page 28.
4. You transmit via a gateway.
   - If you are using the default SYNC gateway, which uses the same protocol as you use for synchronization, you may not need to make any changes to the default gateway settings.
   - If you are using UDP, you may be able to send messages via the default settings.
   - If you are sending SMS notifications, you need to configure gateways and carriers and also specify SMS listening libraries.
   See “Gateways and carriers” on page 20 and “Listening libraries” on page 46.

Other resources for getting started

- Sample applications are installed to samples-dir\MobiLink\SIS_*.*. (For more information about samples-dir, see “Samples directory” [SQL Anywhere Server - Database Administration].)
- You can post questions on the MobiLink newsgroup: sybase.public.sqlanywhere.mobilink
CHAPTER 2

Setting up server-initiated synchronization

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push requests</td>
<td>10</td>
</tr>
<tr>
<td>Setting properties</td>
<td>14</td>
</tr>
<tr>
<td>Notifiers</td>
<td>17</td>
</tr>
<tr>
<td>Gateways and carriers</td>
<td>20</td>
</tr>
<tr>
<td>Device tracking</td>
<td>22</td>
</tr>
<tr>
<td>Listeners</td>
<td>28</td>
</tr>
</tbody>
</table>
Push requests

A push request takes the form of some data that you insert into a table on the MobiLink consolidated database, or in some cases data inserted into a temporary table or even just a SQL result set. You can create push requests in any way that you cause data to be inserted into a table.

The Notifier sends a message to a remote database when it detects a push request. The push request specifies the content of the message, along with when, how, and to whom the message should be sent.

Creating the push request table

A push request is a row in a SQL result set on the consolidated database that contains the following columns in the following order. The first five columns are required and the last two columns are optional. The Notifier uses the request_cursor property to fetch push requests.

In a typical implementation, you add a push request table to your consolidated database. You populate the push request table when a change is detected on your consolidated database, and use the request_cursor Notifier event to send push requests to remote Listeners. The request_cursor Notifier event must receive the following columns in the following order:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request id</td>
<td>INTEGER. A unique ID for a push request.</td>
</tr>
<tr>
<td>gateway</td>
<td>VARCHAR. The gateway on which to send the message. This can be a predefined or</td>
</tr>
<tr>
<td></td>
<td>user-defined gateway. Predefined gateways are Default-DeviceTracker, Default-</td>
</tr>
<tr>
<td></td>
<td>SMTP, and Default-UDP.</td>
</tr>
<tr>
<td>subject</td>
<td>VARCHAR. The subject line of the message.</td>
</tr>
<tr>
<td>content</td>
<td>VARCHAR. The content of the message.</td>
</tr>
<tr>
<td>address</td>
<td>VARCHAR. The destination address. For a SYNC gateway or DeviceTracker gateway,</td>
</tr>
<tr>
<td></td>
<td>it is the MobiLink user name of the Listener, or other MobiLink user names that you</td>
</tr>
<tr>
<td></td>
<td>register using dblsn -t+. For an SMTP gateway without device tracking, it is an email</td>
</tr>
<tr>
<td></td>
<td>address. For a UDP gateway without device tracking, it is an IP address or host name,</td>
</tr>
<tr>
<td></td>
<td>optionally followed by a colon and port number.</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>resend interval</td>
<td>VARCHAR. Optional. How often the message should be resent. The default unit is minutes. You can specify S, M, and H for units of seconds, minutes, and hours. You can also combine units, as in 1H 30M 10S.</td>
</tr>
<tr>
<td></td>
<td>The resend interval is especially useful when the remote device is listening for UDP and the network is unreliable. The Notifier assumes that all attributes associated with a resendable notification request do not change: subsequent updates are ignored after the first poll of the request. The Notifier automatically adjusts the next polling interval if a resendable notification must be sent before the next polling time. You can stop a resendable notification using the request_cursor query or by deleting the request from the request table. The default is to send exactly once, with no resend. Delivery confirmation from the intended Listener may stop a subsequent resend.</td>
</tr>
<tr>
<td>time to live</td>
<td>VARCHAR. Optional. The time until the resend expires. The default unit is minutes. You can specify S, M, and H for units of seconds, minutes, and hours. You can also combine units, as in 1H 30M 10S. If this value is 0, null, or not specified, the default is to send exactly once, with no resend.</td>
</tr>
</tbody>
</table>

**Note:** push requests can also be stored in temporary tables and across multiple tables.

For more information about addressing notifications when you are using device tracking, see “Listener options for device tracking” on page 23.

**Example**

Following is a SQL Anywhere CREATE TABLE statement that creates a push request table.

```sql
create table PushRequest (  
    req_id integer default autoincrement primary key,  
    gateway varchar(128),  
    subject varchar(128),  
    content varchar(128),  
    address varchar(128),  
    resend_minute varchar(30),  
    minute_to_live varchar(30)  
)  
```

The following code uses the ml_add_property stored procedure to create a request_cursor property that creates the push request.

```sql
call ml_add_property( 'SIS', 'Notifier(Simple)', 'request_cursor',  
    'select req_id,  
    gateway,  
    subject,  
    content,  
    address,  
    resend_minute,  
    minute_to_live  
    from PushRequest' );
```
Creating push requests

You can create push requests in any way that you cause data to be inserted into a table. Following is a list of common ways to create push requests:

- Specify SQL synchronization logic in Notifier properties. The most obvious property for creating push requests is the begin_poll property.
  
  A benefit of creating push requests inside the Notifier is that contention is minimized because only one database connection is used for push requests.
  
  See “begin_poll property” on page 59.

- Define a database trigger. For example, create a trigger that detects when a price changes and then inserts push request data into a table of push requests.
  
  See “Introduction to triggers” [SQL Anywhere Server - SQL Usage].

- Use MobiLink synchronization logic to create push requests that notify other MobiLink users. For example, create an end_upload script that detects that a specific change has been uploaded and then creates a push request to update other users who should have the same data.
  
  See “end_upload table event” [MobiLink - Server Administration].

- Use a database client application that inserts data into a push request table directly.

- Manually insert push request data using an Interactive SQL utility.

Sending push requests

The Notifier sends a set of push requests to remote devices by executing a SQL query that you provide in the request_cursor property.

For more information about querying the consolidated database, see “request_cursor property” on page 62.

Deleting push requests

You delete push requests to prevent resending old messages. Deleting requests in a timely manner can help minimize the number of messages sent and increase the efficiency of the application.

The most straightforward way to delete push requests is to use the Notifier property request_delete. This property is a SQL statement with a request ID as a parameter. Using this statement, the Notifier deletes requests that have been confirmed as delivered or that have expired.

For more information, see “request_delete property” on page 62.

Built-in delivery confirmation is not available on Palm devices; on all devices, it can be disabled. You can optionally implement your own delivery-confirmation mechanism. For example, your synchronization logic can delete push requests from a request table when a specific synchronization occurs.
Notifying the Listener with sa_send_udp

SQL Anywhere databases include a system stored procedure called sa_send_udp that can be used to send UDP notifications to the Listener.

If you use sa_send_udp as a way to notify the Listener, you should append a 1 to your UDP packet. This number is a server-initiated synchronization protocol number. In future versions of MobiLink, new protocol versions may cause the Listener to behave differently.

See “sa_send_udp system procedure” [SQL Anywhere Server - SQL Reference].

Example

On a device, start the Listener as follows, where path is the location of your Internet Explorer program:

dblsn -v -l "message=TheMessage;action=start 'path\iexplore.exe' http://www.sybase.com"

On a different device, start a SQL Anywhere database. Start Interactive SQL, and connect to the database. Execute the following SQL. (Note that the UDP packet has a 1 appended to it.)

call SA_SEND_UDP('machine#1_ip_name',5001,'TheMessage1')

Internet Explorer opens, showing the Sybase home page.

To make this example work on a single device, use localhost as the first parameter to sa_send_udp.
Setting properties

Notifiers, gateways, and carriers are configured via properties. These properties can be stored in the
ml_property MobiLink system table or in a Notifier properties file.

Storing properties in the database

You can store property settings in the MobiLink system table ml_property. There are two ways to do this:

- Use the Notification tab in the MobiLink plug-in in Sybase Central.
  For more information, click Help on the Sybase Central Notifier windows.

- Use the stored procedure ml_add_property.
  See “ml_add_property system procedure” [MobiLink - Server Administration].

Storing properties in a properties file

Alternatively, you can store options in a Notifier properties file. This is a text file that you can edit with a
text editor. If you use the properties file, you cannot use the default SYNC gateway. See:

- “Notifier properties file” on page 15
- “SYNC gateway properties” on page 70

Setting properties in more than one place

If you specify properties in both the ml_properties table and the Notifier properties file, the settings are
determined as follows:

1. Server-initiated synchronization properties in the ml_property table in the consolidated database are
   loaded.

2. If a Notifier properties file is specified with the -notifier option, the settings in this file are loaded on top
   of the settings from the database.

   If a Notifier properties file is not specified, and if the default configuration file is found
   (config.notifier), the settings in the default file are loaded on top of the settings from the database.

Changing properties

Properties are read at startup. When you change properties, you must shut down and restart the MobiLink
server for them to take effect.

Properties

For a detailed list of the Notification properties you can set, see:

- “Common properties” on page 54
- “Notifier properties” on page 55
- “Gateway properties” on page 66
- “Carrier properties” on page 74
Notifier properties file

Properties for Notifiers, gateways, and carriers can be stored in the ml_property MobiLink system table or in the Notifier properties file. See “Setting properties” on page 14.

The Notifier properties file is a text file. It can have any name. The easiest way to create this file is to alter the template, template.notifier, located in samples-dir\MobiLink\ (For information about samples-dir, see “Samples directory” [SQL Anywhere Server - Database Administration].)

You can export the properties from the ml_property table into your Notifier properties file. To do this, connect to the MobiLink plug-in in Sybase Central, right-click the Notification folder, and choose Export Settings. The exported file may be copied to a different location and used to easily configure a Notifier there.

You can have several Notifier property files. To identify the properties file you want to use, specify the name and location when you start mlsrv11 with the -notifier option. Following is a partial mlsrv11 command line:

```
mlsrv11 ... -notifier "c:\CarDealer.notifier"
```

For information about how properties are read if you do specify a properties file on the command line, see “Setting properties in more than one place” on page 14.

A Notifier properties file can configure and start multiple Notifiers and multiple gateways. You provide a name for each Notifier and gateway that you want to define.

Notifier properties are normally entered on one line, but you can use the backslash (\) as a line continuation character.

The backslash is also an escape character. You can use the following escape sequences in your property settings:

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>\u0008: backspace, BS</td>
</tr>
<tr>
<td>\t</td>
<td>\u0009: horizontal tab, HT</td>
</tr>
<tr>
<td>\n</td>
<td>\u000a: linefeed, LF</td>
</tr>
<tr>
<td>\f</td>
<td>\u000c: form feed, FF</td>
</tr>
<tr>
<td>\r</td>
<td>\u000d: carriage return, CR</td>
</tr>
<tr>
<td>&quot;</td>
<td>\u0022: double quote, &quot;</td>
</tr>
<tr>
<td>'</td>
<td>\u0027: single quote, '</td>
</tr>
<tr>
<td>\</td>
<td>\u005c: backslash, \</td>
</tr>
<tr>
<td>\uhhhh</td>
<td>Unicode character (hexadecimal)</td>
</tr>
</tbody>
</table>
## Escape sequence

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\xhh</td>
<td>\xhh: ASCII character (hexadecimal)</td>
</tr>
<tr>
<td>\e</td>
<td>\u001b: Escape, ESC</td>
</tr>
</tbody>
</table>

### Example

For a fully commented sample Notifier properties file, see `samples-dir\MobiLink\template.notifier`. (For information about `samples-dir`, see “Samples directory” [SQL Anywhere Server - Database Administration].)

### Note

If you want to use the default SYNC gateway, you cannot store Notifier configuration settings in this properties file. You must store them in the database. See “Setting properties” on page 14.
Notifiers

The Notifier runs on the same computer as the MobiLink server. The Notifier polls the consolidated database on a regular basis, looking for push requests. When it detects a push request, it sends a message to a remote device. It also contains functionality for executing custom SQL scripts, handling delivery confirmation, deleting push requests, and reconnecting after lost database connections. You may use the custom SQL scripts to monitor your data and create push requests.

You can have more than one Notifier running within a single instance of the MobiLink server. Each Notifier keeps one database connection open all the time.

For an example of multiple Notifiers, see the sample located in the samples-dir\MobiLink\SIS_MultipleNotifier. For more information about samples-dir, see “Samples directory” [SQL Anywhere Server - Database Administration].

Starting the Notifier

You start Notifiers on the mlsrv11 command line. To start the Notifier, use the mlsrv11 option -notifier. Optionally, you can also specify the name of your Notifier properties file, if you have one.

Following is a partial mlsrv11 command line:

   mlsrv11 ... -notifier c:\myfirst.notifier

For information about how properties are read if you specify a properties file in the command line, see “Setting properties in more than one place” on page 14.

For information about how properties are applied, see “Setting properties” on page 14.

For more information about the -notifier option, see “-notifier option” [MobiLink - Server Administration].

When you use the -notifier option, you start every Notifier that you have enabled. For more information about enabling Notifiers, see “enable property” on page 63.

Note for QAnywhere users

When you run the MobiLink server, mlsrv11, with the -m option, a Notifier is automatically started for you.

Another way to send notifications

As an alternative to the Notifier, you can also use the sa_send_udp stored procedure to send simple notifications if you are using a SQL Anywhere consolidated database.

See “Notifying the Listener with sa_send_udp” on page 13.
Configuring Notifiers

The Notifier allows you to create custom SQL to program the server-initiated synchronization process. You do this by setting properties. For example, you would configure properties to perform tasks such as the following:

- Create push requests in response to changes in the consolidated database. The `begin_poll` property is often used in this way.

- Use the `request_cursor` property to determine what information is sent in a notification, to whom, where, and when. The Notifier uses the result set returned by the `request_cursor` to send push requests to remote Listeners.

  **Note**: The `request_cursor` property is the only required property. See “`request_cursor` property” on page 62.

- Delete push requests with the `request_delete` property.

- Set a polling interval using the `poll_every` property.

For a complete list of Notifier properties, see “MobiLink Notifier properties” on page 53.

For information about how to set Notifier properties, see “Setting properties” on page 14.

Notifier property sequence

The following pseudo-code shows the sequence in which server-initiated synchronization properties are used. Note that except for `request_cursor`, all of these properties are optional.

```sql
connect_string
isolation
begin_connection
poll_every
For each poll (begin_poll
  begin_poll
  shutdown_query
  request_cursor
  For all requests expired before required confirmation (error_handler
    request_delete
  )
  end_poll
) end_poll
end_connection
```

Notifiers in a MobiLink server farm

Prior to version 11.0, server-initiated synchronization could cause redundant notifications and significant load on the consolidated database in a MobiLink server farm environment. Now you can run a Notifier on every MobiLink server in the farm and the Notifiers together ensure that there are no redundant notifications to the same Listener. The `-lsc` server option is used to pass information to other servers when they want to connect to the local MobiLink server. See “-lsc option” [MobiLink - Server Administration].

This feature makes one Notifier the primary, and all other Notifiers in the farm secondary. The primary Notifier is in control of notification, either directly or indirectly via the secondaries. The secondary Notifiers
also route Listener information to the Primary Notifier, so it knows where the Listeners are and how to reach them.

If the MobiLink server running the primary Notifier goes down, the server farm fails over and elects a new primary Notifier, and notifications continue.

Listeners may connect to any ML server in the farm without needing to know which is the primary server.

In order to use this feature, the following mlsrv11 command line options are required on all MobiLink servers in the farm:

- “-lsc option” [MobiLink - Server Administration]
- “-notifier option” [MobiLink - Server Administration]
- “-zs option” [MobiLink - Server Administration]

**Example**

On host001:

```
mlsrv11 -notifier -zs ml001 -lsc tcpip(host=host001;port=2439) ...
```

On host007:

```
mlsrv11 -notifier -zs ml007 -lsc tcpip(host=host007;port=2439) ...
```
Gateways and carriers

Gateways are the mechanisms for sending messages. You can define SYNC gateways, UDP gateways, and SMTP gateways. In most cases, you can also use a device tracker gateway that automatically decides which gateway to use.

- **Device tracker gateway**  When you use the device tracker gateway, the address in your push request is the MobiLink user name of the Listener.

  Device tracking allows automatic management of remote device address changes. When using device tracking, the SYNC gateway is attempted first, with fallback to the UDP and then the SMTP gateways if they are enabled.

  Use of the device tracker gateway is recommended. If you do not use device tracking, your request_cursor must include a UDP or SMTP gateway name and address, and, for each push request, only that gateway becomes tried.

  See “Device tracking” on page 22 and “Device tracker gateway properties” on page 66.

- **SYNC gateway**  The SYNC gateway can use the same protocol as you use for synchronization. The connection is persistent, and is used for all Listener communication (notifications, confirmation, and device tracking). If you are using SYNC, you may not need to make any changes to the default gateway settings.

  See “SYNC gateway properties” on page 70.

- **UDP gateway**  This gateway allows you to send push requests to remote listeners using UDP. If you are using UDP, you may not need to make any changes to the default gateway settings.

  See “UDP gateway properties” on page 71.

- **SMTP gateway**  You can use the SMTP gateway to send messages to SMS listeners via a wireless carrier’s email-to-SMS service. For SMTP, you need to configure an SMTP gateway and carrier.

  See “SMTP gateway properties” on page 67.

When you use an SMTP gateway, you configure carriers to store information about the public wireless carriers that you want to use. Carrier information is used to create SMS email addresses from device tracking information that is sent up from Listeners.

Configuring gateways and carriers

For information about how to set properties for gateways and carriers, see “Setting properties” on page 14.

For a detailed list of gateway and carrier properties, see:

- “Device tracker gateway properties” on page 66
- “UDP gateway properties” on page 71
- “SMTP gateway properties” on page 67
- “Carrier properties” on page 74
Gateways

There are four default gateways. They are installed when you run the MobiLink setup scripts for your consolidated database. The default gateways are called:

- Default-DeviceTracker gateway
- Default-SYNC gateway
- Default-UDP gateway
- Default-SMTP gateway

A device tracker gateway can have up to three subordinate gateways: one SYNC, one SMTP, and one UDP. The device tracker gateway automatically routes each message to one of its subordinate gateways based on device tracking information sent up from Listeners. See “Device tracking” on page 22.

Default-SYNC, Default-UDP, and Default-SMTP are preconfigured with some settings that may work out of the box, especially SYNC and UDP. In most cases, you should use the default gateways. You can customize their configuration, if required.

You should not delete the default gateways or change their names. You can create additional gateways and assign names to them.

Carriers

You only need to configure a carrier if you are using device tracking with an SMTP gateway. Carrier configuration allows you to specify information such as the name of a network provider, their email prefix, network provider id, and so on. This information is necessary for the Notifier to construct email addresses for each public wireless carrier's email-to-SMS service.

To configure a carrier, you can run the Listener on a device that has a modem and service provider working, and inspect the Listener messages window or message log file. If your Listener uses the -x option to connect to a running MobiLink server, you can also find carrier device tracking information in the ml_device_address MobiLink system table.

Once a carrier is configured, it requires no further attention. The carrier can be used to send SMS messages via SMTP to all devices using that public wireless carrier.

See “Carrier properties” on page 74.
Device tracking

Device tracking allows you to address a remote database by supplying only its MobiLink user name in a push request. When device tracking is enabled, MobiLink keeps track of how to reach users. For example with an SMTP gateway, when a device's IP address changes, the Listener synchronizes with the consolidated database to update the device tracking information in the MobiLink system table ml_device_address. The device tracker gateway first attempts to use a SYNC gateway, and if the delivery fails it then attempts using a UDP gateway or an SMTP gateway.

In most cases, you should be able to use device tracking. It is recommended that you use it because it simplifies deployment.

Most 9.0.1 or later Listeners support device tracking. If you are using Listeners that don't support device tracking, you can still use a device tracker gateway by providing tracking information yourself.

See “Using device tracking with Listeners that don't support it” on page 24.

If you do not use device tracking, your request_cursor must include a specific UDP or SMTP gateway name and address. For each push request, only that gateway becomes used, and no other gateway is attempted.

Setting up device tracking

To set up device tracking

1. Set up a UDP gateway and/or SMTP gateway, if necessary. Note: Typically, the UDP gateway is usable without further configuration, and you do not have to set up a gateway or carrier for it. However, if you want to use email-to-SMS notification, the default SMTP gateway requires configuration.

   See “Configuring gateways and carriers” on page 20.

2. Your request_cursor script should have the following settings:
   ● The gateway name must be the name of a device tracker gateway. The default instance is called Default-DeviceTracker.
   ● The address must be a MobiLink user name. By default, it can be the Listener user name. However, you can use the dblsn option -t+ to add the MobiLink user name of the remote database that you are synchronizing, and then directly address that database.

   See “request_cursor property” on page 62.

3. Add the Listener name to the MobiLink ml_user system table.

   The default Listener name is device_name-dblsn, where device_name is the name of your device. You can find the device name in your Listener messages window. Optionally, you can set the device name using the dblsn -e option. You can specify a different Listener name using the dblsn -u option.

   Whether or not you use the default name, you may need to add the Listener_name to the ml_user MobiLink system table on your consolidated database. This is because the Listener_name is a MobiLink user name. Like other MobiLink user names, it must be unique and it must be added to the ml_user MobiLink system table on your consolidated database.
See “Creating and registering MobiLink users” [MobiLink - Client Administration].

4. Start the Listener with the required options.

See “Listener options for device tracking” on page 23.

Listener options for device tracking

The following dblsn options are used for device tracking.

Use -x, -u, and -w to specify how to connect to the MobiLink server. This is required if you are using device tracking so that the remote device can update the consolidated database if the address changes. These are also required if you want to send delivery confirmations to the consolidated database.

The -t+ option is recommended. With it, you can register the MobiLink user name of your remote database and use it when you address notifications instead of addressing the MobiLink user name of the Listener database. You only need to do this once.

- **-t+ ml_user** Use this option to register the MobiLink user name of your remote database so that you can use that user name as the address in your push requests.

   You can register multiple MobiLink user names with -t+. This is useful if you need to address notifications to different applications on the remote device, such as multiple remote databases.

   This mapping is retained on the server (in the ml_listening table) once tracking information is uploaded successfully, so you only need to register a MobiLink user name once unless you change the MobiLink user name or location. However, using -t+ multiple times is not harmful.

- **-t- ml_user** To disable a MobiLink user name that was registered for device tracking with -t+, use -t-.

- **-u ML_user_name** Use -u to create a MobiLink user name for the Listener. The -u option is optional because there is a default Listener_name, which is device_name-dblsn, where device_name is the name of your device. You can find the device name in your Listener messages window. Optionally, you can set the device name using the -e option.

   Whether or not you use the default name, you may need to add the Listener_name to the ml_user MobiLink system table on your consolidated database. This is because the Listener_name is a MobiLink user name. Like other MobiLink user names, it must be unique and it must be in the ml_user MobiLink system table on your consolidated database.

See “Creating and registering MobiLink users” [MobiLink - Client Administration].

- **-w password** This option sets the password for the Listener name.

- **-x connection-parameters** Use -x to specify how to connect to the MobiLink server. This is required if you are using device tracking because it lets the remote device update the consolidated database if the address changes. This option is also required if you want to send delivery confirmations to the consolidated database.

- **-y** This option updates the password for the Listener name.

For more information about Listener options, see “Listener syntax” on page 38.
Example

The following command starts the Listener with device tracking.

```
dblsn -l "subject=sync;action='run dbmlsync.exe -c dsn=rem1'"
-x tcip(host=MLSERVER_MACHINE) -t+ user1 -u remoteuser1
```

Stopping device tracking

It might be useful to stop device tracking in situations such as the following:

- Your device listens only on UDP on a static IP address.
- Your device listens only on UDP and has dynamic IP with low latency DNS update, so you can use a static IP name to address your device directly.

To stop device tracking when you want to continue using delivery confirmation, use the dblsn option -g.

For more information about dblsn options, see “Listener syntax” on page 38.

Using device tracking with Listeners that don't support it

You cannot use the completely automatic form of device tracking if any of your Listeners have the following characteristics:

- are prior to Adaptive Server Anywhere 9.0.1 or are Palm Listeners
  
  For information about how to set up device tracking in these situations, see “Manually setting up device tracking” on page 24.
- are listening on UDP, and remote IP addresses are unreachable from the MobiLink server machine
  
  For information about how to deal with this situation, see “Unreachable addresses” on page 26.

Manually setting up device tracking

Several stored procedures are provided to help you manually set up device tracking for 9.0.0 Listeners or Palm Listeners. These stored procedures manipulate the MobiLink system tables ml_device, ml_device_address, and ml_listening on the consolidated database. With manual device tracking, you can address recipients by MobiLink user name—without providing network address information—but the information cannot be automatically updated by MobiLink if it changes: you must change it yourself.

This method is especially useful for SMTP gateways because email addresses don't tend to change. For UDP gateways, it is more difficult to rely on static entries if your IP address changes every time you reconnect. You may get around this problem by addressing by host name instead of IP address, but in that case slow updates to DNS server tables can cause misdirected messages. You can also deal with changing IP addresses by setting up the following stored procedures to update the MobiLink system tables programmatically.

To manually set up device tracking for 9.0.0 Listeners or Palm Listeners

1. For each remote device, add a device record to the ml_device MobiLink system table. For example,

```sql
call ml_set_device('myFirstTreo180',
```
The first parameter, myFirstTreo180, is a user-defined unique device name. The second parameter contains optional remarks about the Listener version. The third parameter, set here to 1, specifies a Listener version; use 0 for Listeners from SQL Anywhere 9.0.0, 1 for post-9.0.0 Palm Listeners, and 2 for post-9.0.0 Windows Listeners. The fourth parameter specifies optional device information. The fifth parameter is set to y here, which specifies that device tracking should be ignored; if this were set to n, device tracking would overwrite this record. The final parameter contains optional remarks on the source of this record.

See “ml_set_device” on page 83.

2. For each device that you just added, add an address record to the ml_device_address MobiLink system table. For example,

```sql
call ml_set_device_address(
    'myFirstTreo180',
    'ROGERS AT&T',
    '3211234567',
    'y',
    'y',
    'manually entered by administrator' );
```

The first parameter, myFristTreo180, is a user-defined unique device name. The second parameter is a network provider ID, and must match a carrier's network_provider_id property. (See “network_provider_id property” on page 74.) The third parameter is an IP address for UDP or the phone number of your SMS-capable device. The fourth parameter, set here to y, activates this record for sending notifications. The fifth parameter, set here to y, specifies that device tracking should be ignored; if this were set to n, device tracking could overwrite this record. The final parameter contains optional remarks on the source of this record.

For information about how to locate carrier information, see “Device tracking” on page 22.

For more information about using ml_set_device_address, see “ml_set_device_address” on page 85.

3. For each remote database, add a recipient record to the ml_listening MobiLink system table for the device that was just added. This maps the device to the MobiLink user name. For example,

```sql
call ml_set_listening(
    'myULDB',
    'myFirstTreo180',
    'y',
    'y',
    'manually entered by administrator' );
```

The first parameter is a MobiLink user name. The second parameter is a user-defined unique device name. The third parameter, set here to y, activates this record for device tracking addressing. The fourth parameter, set here to y, specifies that device tracking should be ignored; if this were set to n, device tracking could overwrite this record. The final parameter contains optional remarks on the source of this record.

See “ml_set_listening” on page 87.
Troubleshooting gateways

This section describes some known problems and solutions connected with communication between remote devices and servers.

Unreachable addresses

Symptom

The Notifier cannot reach the device with the tracked IP address.

Cause

Some or all devices cannot be addressed directly because they are private relative to the MobiLink server. For example, a remote device is on a private sub-network and its address is internal to that network.

Remedy

Try one of the following:

- If the IP address is assigned by a public wireless carrier or ISP, you may be able to upgrade your carrier plan so that you can obtain public IP addresses instead of private ones.
- If you are using Wi-Fi, the IP security policy in your organization may stop your device from being reachable. Contact your IT department for assistance.
- Use an SMTP gateway.

If the device's IP address is never reachable, you may want to stop device tracking on the Listener with the -g option. The -g option is useful when you do not want to use device tracking but you do want delivery confirmation. If you are using delivery confirmation, the first attempt to connect is through UDP, and the lack of confirmation prevents further UDP attempts.

For more information about delivery confirmation, see “confirmation_handler property” on page 56.

Tracked address is not correct

Symptom

Device tracking is not picking the best IP address for a device.

Cause

There may be a problem with the routing table on the device.

Remedy

Try one of the following:

- Fix the routing table.
• Use the ml_set_device_address stored procedure to ignore tracking for the device and set the address parameter to the correct address. Be sure to set the fourth parameter to y. In addition, use -g for the problematic Listener.

See “ml_set_device_address” on page 85.
Listeners

The Listener runs on remote devices. It receives messages from the Notifier and processes them into actions based on message handlers that you create. A typical message handler contains filters, actions, and options.

For example, for the following Listener command line, the Listener starts dbmlsync if it receives a message with the subject FullSync:

```
  dblsn -l "subject='FullSync';action='run dbmlsync.exe ...'
```

Following are some of the actions that you can invoke. Typically, the action you choose is synchronization initiated via either dbmlsync or an UltraLite application.

- Start a process.
- Run a process until it completes.
- Post a window message to a process that is already running.
- Perform text-based communication with local or remote applications via TCP/IP with optional confirmation.

Actions can be parameterized with variables derived from the message. This provides extra flexibility in implementing dynamic options.

Normally, you only need to start up one Listener on a device. One Listener can listen on multiple channels and it can serve multiple MobiLink users on the same device. A running Listener always listens on UDP (except for Palm Listeners).

Listeners can also synchronize device tracking information back to the consolidated database. For more information, see “Device tracking” on page 22.

See also

- For Listener syntax and options, see “Listener syntax” on page 38.
- For information about Palm devices, see “Listeners for Palm devices” on page 49.
- For dbmlsync options, see “MobiLink SQL Anywhere client utility [dbmlsync]” [MobiLink - Client Administration].
- For more information about message handlers, see “Message handlers” on page 29.
- Instead of entering dblsn options at a command prompt, it is often convenient to store them in a text file. For more information, see “Storing Listener options” on page 33.

Example

The following command starts the Listener utility. It must be typed on one line.

```
  dblsn -v2 -m -ot dblsn.log -x "host=localhost"
  -l "subject=sync;action='start dbmlsync.exe
  -c eng=rem1;uid=DBA;pwd=sql -ot dbmlsyncOut.txt -k';"
```

The options used in this example are:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-v2</td>
<td>Set verbosity to level 2 (log Listener DLL messages and action tracing).</td>
</tr>
<tr>
<td>-m</td>
<td>Log notification messages.</td>
</tr>
<tr>
<td>-ot</td>
<td>Truncate the log file and send output to it. In this case, the output file is dblsn.log.</td>
</tr>
<tr>
<td>-x</td>
<td>Specify a way to connect to the MobiLink server. This is required for device tracking and delivery confirmation. In this simple example, the only protocol options that are specified are &quot;host=localhost&quot;. For a complete list of protocol options, see “-x option” [MobiLink - Client Administration].</td>
</tr>
<tr>
<td>-l</td>
<td>Specify a message handler. In this case the filter is that a message must contain the subject \texttt{sync}, and the action is to start dbmlsync. Three dbmlsync command line options are also provided: -c specifies a connection string to the MobiLink server for the synchronization; -ot names an output log file; and -k shuts down dbmlsync when the synchronization is complete.</td>
</tr>
</tbody>
</table>

**Message handlers**

Using the dblsn command line, you create message handlers to tell the Listener which messages to filter and what actions should result from each accepted message.

See “Listener utility” on page 37.

**Message interpretation**

Messages (push requests) arrive as a single piece of text with the following structure:

\[
\text{message control_information}
\]

The control_information is for internal use only and is removed prior to message handling. The Listener substitutes non-printable characters with tildes, and then interprets the message portion with the following pattern:

\[
\text{message} = \text{sender} \ subj-open \ subject \ subj-close \ content
\]

\[
\text{subj-open} = ( | [ | { | < | ' | ”}
\]

The subj-open character is determined by the first possible character found by scanning from left to right. The value of subj-open determines the value of subj-close. The possible values of subj-close are ), ], }, >, ' and ".

The location of the first subj-close character marks the end of the subject and the beginning of the content.

The sender is empty when the message begins with a subj-open. In that case, the sender of the message is determined in a delivery path-dependent way. For example, messages going through UDP gateways arrive as [subject] content, and the sender is the IP address. SMTP gateways send an email message that is converted by an email to SMS service into a format that varies between different public wireless carriers.
Filters and action variables

The Listener provides filters and action variables to determine how to process an incoming message (push request).

- **Filters** Filters allow the Listener to determine what action to take in response to the subject, content, and other parts of a message (push request). Filters are specified using the dbln -l option.

  See “Using subject and content filters” on page 30 and “Using the filters message, message_start, and sender” on page 31.

- **Action variables** Action variables allow you to incorporate parts of a message (push request) in the Listener action.

  See “Action variables” on page 44.

Using subject and content filters

Use the filters **subject** and/or **content** to filter messages by subject and/or content as specified in your push requests. When you use these filters, the Listener automatically adjusts the filter to match the format received by the carrier. For example, you may want to filter a message with the subject Sync and the content Orders. You do not have to worry that in UDP, this would appear as [Sync]Orders, and on one email to SMS conversion service, it would be Bob@mail.com[Sync]Orders.

Your subject cannot contain the closing character that is used to enclose the subject. In the previous example, UDP encloses the subject Sync in square brackets. This means that you cannot use a closing square bracket in subjects that might be received over UDP. For SMTP messages, your carrier determines the character used to enclose the subject. This might be one of ), }, >, ‘, ”.

**Note**
For best results, only use alphanumeric characters in your subject when creating push requests.

For SMS messages, the Listener trims leading and trailing spaces, as well as leading and trailing tilde (~) characters, from the sender name, subject, and content. Non-printable characters such as the new line character are deleted by the Listener before filtering.

Filtering by remote ID

For SQL Anywhere remote databases, the first time you synchronize, a **remote ID file** is created that contains the remote ID for the remote database. This file has the same name as the database, but with the extension .rid, and is stored in the same directory as the database.

For UltraLite databases, to specify the remote ID file you use the name of the database.

To use the remote ID in a filter, you must use the dbln -r option and provide the name and location of the remote ID file. Then you can use the $remote_id variable in your filter. If you use -r more than once, the $remote_id refers to the file specified in the -r option just before it.
You can also use the remote ID directly in filters. However, by default the remote ID is a GUID, so unless you provide a more meaningful remote ID name, it is not easy to reference directly.

**Example**

For example, the following code is a partial dblsn command file. It assumes that you have two databases on the device, a SQL Anywhere database called *business.db* and an UltraLite database called *personal.udb*. In this example, ulpersonal is the window class name of the UltraLite application.

```bash
-r "c:\app\db\business.rid"
-l "subject=$remote_id;action='dbmlsync.exe -k -c dsn=business';"
-r "c:\ulapp\personal.udb"
-l "subject=$remote_id;action=post dbas_synchronize to ulpersonal;"
```

**See also**

- `-r` option in “Listener syntax” on page 38
- `$remote_id` variable in “Action variables” on page 44
- “Remote IDs” [*MobiLink - Client Administration*]

### Using the filters message, message_start, and sender

The recommended filters are called **subject** and **content**. However, there are three other types of filter that you may also want to use.

The Listener translates non-printable characters in a message to a tilde (~) so if there are non-printable characters, the filter must also use tildes.

- **message** compares the entire message to text you specify. To match, this filter must also be the exact same length as the message. You can specify only one message per message handler.

  The format of messages is carrier-dependent, and you must account for this if you use the `message`, `message_start`, or `sender` filters. For example, you may want to match a message from a sender named Bob@mail.com with the subject Help and the message Me. In UDP, this would appear as [Help]Me. On Bell Mobility's email to SMS conversion service, it would be Bob@mail.com[Help]Me. On Fido's email to SMS conversion service, it would arrive as Bob@mail.com\n(Help)\nMe, but would be translated by the Listener to Bob@mail.com(Help)Me. You must test with your carrier to determine the appropriate format, using the dblsn options `-v` and `-m`.

- **message_start** compares a portion of the message (from the beginning) to text that you specify. When you specify message_start, the Listener creates the action variables `$message_start` and `$message_end`. See “Action variables” on page 44. There is a maximum of one message_start per message handler.

- **sender** is the sender of the message. You can only specify one sender per message handler. For UDP gateways, the sender is the IP address of the host of the gateway. For SMS email, the sender is the email address embedded in the beginning of the message if the SMS format is compatible with server-initiated synchronization. Otherwise, the sender information is not available.

**Multiple message handlers may be required**

Subject and content are the recommended filters when messages arrive in a compatible format. However, if your message format is incompatible, you need to use the message, message_start, and/or sender filters. In
that case, if the delivery path can vary (sometimes through UDP and sometimes through SMTP), then you
need multiple handlers with different filters.

Connection-initiated synchronization

In addition to initiating synchronization from the server, on Windows devices you can also initiate
synchronization when connectivity changes. This is possible because the Windows Listener generates an
internal message with the content _IP_CHANGED_ whenever there is a change in connectivity, and it
generates an internal message with the content _BEST_IP_CHANGED_ whenever there is a new "best" IP
connection.

The internal messages _IP_CHANGED_ and _BEST_IP_CHANGED_ are generated only on Windows
devices, including Windows Mobile.

Identifying a change in the optimum path to a MobiLink server

An IP connection is considered to be "best" if it is the best connection to use when connecting to the MobiLink
server that is specified with the dblsn -x option. Although the "best" designation is defined by the path to
the MobiLink server, in practice it tends to indicate the best IP connection to use in general.

To make use of a change in the best IP connection, use the keyword _BEST_IP_CHANGED_ in your
message filter. A MobiLink server is required as a destination for the network to determine which route is
optimal, so you must also specify connection parameters for a MobiLink server using the -x option. The
message filter should be of the form:

```bash
-l "message='BEST_IP_CHANGED_';action='..."
```

The $best_ip action variable is very useful with the _BEST_IP_CHANGED_ filter. The value of $best_ip
is the local IP address that represents the best IP connection. If there is no IP connection, $best_ip has the
value 0.0.0.0.

You can only use _BEST_IP_CHANGED_ when the Listener is run on a separate machine from the
MobiLink server.

In the following example, the _BEST_IP_CHANGED_ filter is used to initiate a synchronization when the
best IP connection changes. If the connection is lost, an error is generated.

```bash
dblsn -x http(host=mlserver.company.com)
   -v2 -m -i 3 -ot dblsn.log
   -l "message='BEST_IP_CHANGED_';
        action='start dbmlsync.exe -ra -c eng=remote;uid=DBA;pwd=sql
               -n test_pub'"
```

Identifying any change in connectivity

To make use of a change in IP connectivity on your remote device, use the keyword _IP_CHANGED_ in
your message filter. _IP_CHANGED_ only indicates that there has been a change in IP connectivity. The
message filter should be of the form:

```bash
-l "message='IP_CHANGED_';action='..."
```
The following example shows a message handler that can be used in the dblsn command line. The filter captures messages that contain the content _IP_CHANGED_. The action makes use of the action variables $adapters and $network_names. If the connection is lost, an error is generated.

```plaintext
-l "message=_IP_CHANGED_; action='socket port=12345; sendText=IP changed: $adapters|$network_names; recvText=beeperAck; timeout=5'; continue=yes;"
```

See also

- “Listener utility” on page 37
- “Action variables” on page 44

**Multi-channel listening**

To listen on multiple media, you can start the Listener with the -d option. A library for UDP listening is always loaded by default, but there are several others that you can load. See “Listener syntax” on page 38 and “Listening libraries” on page 46.

For more information about the Listener, see “Listener utility” on page 37.

**Storing Listener options**

A convenient way to configure the Listener is to store the command line options in a configuration file and access it with the @ symbol. The configuration file is a text file. For example, store the settings in `mydblsn.txt` and start the Listener by entering:

```plaintext
dblsn @mydblsn.txt
```

The path to the configuration file must be fully qualified.

See “Using configuration files” [SQL Anywhere Server - Database Administration].

If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file.

See “File Hiding utility (dbfhide)” [SQL Anywhere Server - Database Administration].

You can also store command line options in an environment variable, and call it in the dblsn command line by entering @ and the environment variable name; for example, `dblsn @dblnoptions`. If you have both a file name and an environment variable with the same name, the environment variable is used.

**Default parameters file dblsn.txt**

If you enter dblsn without any parameters, dblsn uses `dblsn.txt` as the default argument file. This feature is particularly useful for CE devices.

Following is a sample parameters file.
#---- SIS_SimpleListener\dblsn.txt
#--------------------------------------------
# This is the default argument file for dblsn.exe
#
#---------------------------------------------------------------------------------
# Device name
# -e device1
#
#---------------------------------------------------------------------------------
# MobiLink connection parameters
# -x host=localhost
#
#---------------------------------------------------------------------------------
# Verbosity level 2
# -v2
#
#---------------------------------------------------------------------------------
# Show notification messages in the MobiLink server messages window and
# message log file
# -m
#
#---------------------------------------------------------------------------------
# Polling interval of 1 seconds
# -i 1
#
#---------------------------------------------------------------------------------
# Truncate, then write output to dblsn.log
# -ot dblsn.log
#
#---------------------------------------------------------------------------------
# First message handler
# - No filter, so it applies to all messages
# - Try to send the message to the beeper utility
# - If that fails, start the beeper utility with the message
# - Message handling continues with the next handler
# -l "action='socket port=12345;
# sendText=$sender:$message;
# recvText=beeperAck;
# timeout=5';
# altaction='start java.exe Beeper 12345 $sender:$message';
# continue=yes;"
#
#---------------------------------------------------------------------------------
# Second message handler
# - Only applies to messages with subject equals 'shutdown'
# - The action is to send "shutdown" to the beeper utility
# - Message handling continues with the next handler
-l "subject='shutdown';
   action='socket port=12345;
   sendText=shutdown;
   recvText=beeperAck;
   timeout=5';
   continue=yes;"

# Third handler
# - Only applies to messages with subject equals 'shutdown'
# - The action is to shut down the MobilLink Listener
# -
-1 "subject='shutdown';
   action='DBLSN FULL SHUTDOWN';"
CHAPTER 3

Listener utility

Contents

Listener syntax ........................................................................................................... 38
Listener utility

Listener syntax

The Listener utility, dblsn, configures and starts the Listener on Windows devices, including Windows Mobile.

This section is a detailed reference of the Listener utility. For usage information, see “Listeners” on page 28.

For information about Palm devices, see “Listeners for Palm devices” on page 49.

Syntax

dblsn [ options ] -l message-handler [ -l message-handler... ]

message-handler :
[ filter;... ]action
[ ;continue = yes ]
[ ;maydial = no ]
[ ;confirm_delivery = no ]

filter :
[ subject = string ]
[ content = string ]
[ message = string | message_start = string ]
[ sender = string ]

action :
action = command [ ; altaction = command ]

command :
start program [ program-arguments ]
run program [ program-arguments ]
post window-message to { window-class-name | window-title }
tcpip-socket-action
| DBLSN FULL SHUTDOWN

tcpip-socket-action :
socket port=app-port
[ ;host=app-host ]
[ ;sendText=text1 ]
[ ;recvText=text2 [ ;timeout=num-sec ] ]

window-message : string | message-id

Parameters

Options The following options can be used to configure the Listener. They are all optional.

<table>
<thead>
<tr>
<th>dblsn options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>Reads options from the specified environment variable or configuration file. If both exist, the environment variable is used. See “Storing Listener options” on page 33.</td>
</tr>
<tr>
<td>dblsn options</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>-a option</td>
<td>Specifies a Listener DLL option. If you specify multiple -d options, each -a is for the -d option it follows. To specify multiple options, repeat -a. For example, -a port=2439 -a ShowSenderPort. To see options for your dll, enter dblsn -d filename.dll -a ? or see “Listening libraries” on page 46.</td>
</tr>
<tr>
<td>-d filename</td>
<td>Specifies the Listener dll that you want to use. The default dll is lsn_udp.dll. For SMTP gateways, there are several dll's that you can specify. For a list, see “Listening libraries” on page 46. To enable multi-channel listening, specify multiple dlls by repeating -d. After each -d option, specify the -a and -i options that relate to the dll. For example, dblsn.exe -d lsn_udp.dll -i 10 -d maac750.dll -i 60</td>
</tr>
<tr>
<td>-e device-name</td>
<td>Specifies the device name. By default, the device name is automatically extracted from the system. If you do not use -e, you must ensure that all devices have unique names.</td>
</tr>
<tr>
<td>-f string</td>
<td>Specifies extra information about the device. By default, this information is the operating system version. Using this option overrides the default value.</td>
</tr>
<tr>
<td>-ga</td>
<td>Turns on asynchronous ip tracking. This option has been deprecated. Asynchronous ip tracking is now implicit.</td>
</tr>
<tr>
<td>-gi</td>
<td>Controls the ip tracker polling interval. The default is 60 seconds. This option is ignored if -ga is specified. This polling mechanism can be used for trouble shooting, however is no longer needed under normal circumstances.</td>
</tr>
<tr>
<td>-i seconds</td>
<td>Sets the polling interval in seconds for SMTP connections. This is the frequency at which the Listener checks for messages. If you use multiple -d options, each -i setting is for the -d it follows. The default for SMTP is 30 seconds. For UDP connections, the Listener attempts to connect immediately.</td>
</tr>
<tr>
<td>-m</td>
<td>Turns on message logging. The default is off.</td>
</tr>
<tr>
<td>-ni</td>
<td>Stop tracking UDP addresses when -x is used. This is useful when you want device tracking to exclude UDP address updates due to IP address changes.</td>
</tr>
<tr>
<td>dblnsn options</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>-np</td>
<td>Use to disable persistent connection.</td>
</tr>
<tr>
<td>-ns</td>
<td>For Windows Mobile 2003 and later Phone Edition, the Listener listens for SMS as well as UDP. It uses a filtering mechanism that runs as a system process, so the filtering continues even when the Listener is not running. This option disables this behavior. When you use -ns, the Listener listens by default on UDP only, and you can use SMS listening by specifying a listening library with the -d option.</td>
</tr>
<tr>
<td>-nu</td>
<td>Disables default UDP listening.</td>
</tr>
<tr>
<td>-o filename</td>
<td>Logs output to a file. If -o is not used, output is logged to the Listener messages window.</td>
</tr>
<tr>
<td>-os bytes</td>
<td>Specifies a maximum size for the log file in bytes. The minimum size is 10000. By default, there is no limit.</td>
</tr>
<tr>
<td>-ot filename</td>
<td>Logs output to file, but deletes the contents first.</td>
</tr>
<tr>
<td>-p</td>
<td>Allows automatic idle power-off. This option has an effect only on CE devices. Use it to allow the device to shut down when idle. By default, the Listener prevents the device from shutting itself down so that Listening may continue.</td>
</tr>
<tr>
<td>-pc {+</td>
<td>-}</td>
</tr>
<tr>
<td>-q</td>
<td>Runs in a minimized window.</td>
</tr>
<tr>
<td>-r remote-id-file</td>
<td>Identifies a MobiLink remote database that is involved in the responding action of a message handler. When -r is used, the $remote_id variable can be used in message handlers to refer to the remote ID that is contained in remote-id-file. This option can simplify references to remote IDs, which by default are GUIDs. If you have multiple databases on the device, you can use this option multiple times. The remote-id-file is the full path/name of a file that contains the remote ID. This file is automatically created by dbmlsync after the first synchronization. It uses the same location and name as the database file, and has the extension .rid. For UltraLite databases, use the UltraLite database name as the remote ID file. See “Filtering by remote ID” on page 30.</td>
</tr>
</tbody>
</table>
### dblsn options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-t {+-} ml-user-alias</td>
<td>Register remote databases for notification so that you can address the remote database by name when using device tracking. You can also use the $remote_id variable to identify databases. See “Listener options for device tracking” on page 23 and “Action variables” on page 44.</td>
</tr>
</tbody>
</table>
| -u Listener-name | Specifies a MobiLink user name. This name is used when the Listener needs to connect to the MobiLink server, which it does for device tracking, confirmations, and persistent connection.  

The default MobiLink user name is *device-name*-dblsn.  

MobiLink user names must be registered with the MobiLink server. See “Adding MobiLink user names to the consolidated database” [MobiLink - Client Administration]. |
| -v \[ level \] | Sets the verbosity level for the dblsn message log. The *level* can be 0, 1, 2, or 3:  

- 0 - show no informational messages (the default).  
- 1 - show Listener dll messages, basic action tracing steps, and command line options.  
- 2 - show level 1 plus detailed action tracing steps.  
- 3 - show level 2 plus polling and listening states.  

To output notification messages, you must also use -m (see above). See “Logging database server messages to a file” [SQL Anywhere Server - Database Administration]. |
| -w password | Specifies a password for the Listener-name. See “Listener options for device tracking” on page 23. |
| -x \{http|https|tcpip\} \[(key-word=value;...)] | Specifies the network protocol and protocol options for the MobiLink server. For a list of protocol options, see “MobiLink client network protocol options” [MobiLink - Client Administration]. A connection to the MobiLink server is required for the Listener to send device tracking information and delivery confirmation to the consolidated database, and for the SYNC gateway. See “Listener options for device tracking” on page 23. |
| -y new-password | Specifies a new MobiLink password for the Listener name. If your authentication system allows remote devices to change their passwords, this option lets them send up the new password. See “Listener options for device tracking” on page 23. |

### Message-handlers

The -l option allows you to specify a message handler, which is a filter-action pair. The filter determines which messages should be handled, and the action is invoked when the filter matches a message.
You can specify multiple instances of -l. Each instance of -l specifies a different message handler for each incoming message. Message handlers are processed in the order they are specified.

You can also specify the following options for message handlers:

- **continue=yes** specifies that the Listener should continue after finding the first match. This is useful when you specify multiple -l clauses to cause one message to initiate multiple actions. The default is no.

- **maydial=no** specifies that the action cannot dial the modem. This provides information to the Listener to decide whether to release the modem or not before the action. This option is useful when the action or altaction need exclusive access to the modem used by the Listener. The default is yes.

- **confirm_delivery=no** specifies that this handler should not confirm delivery. A message requires confirmation if the gateway used to send it has its confirm_delivery property set to yes. Delivery can only be confirmed if the message requires confirmation and if the handler accepts the message. The default is yes.

  Normally, you do not need to specify this option. By default the first handler that accepts the message sends delivery confirmation, if required. This option can be used when multiple handlers might accept the same message to give you finer control over which handler should confirm the delivery.

  For information about handling delivery confirmation on the server, see “confirmation_handler property” on page 56.

**Filters**

You specify a filter to compare to an incoming message. If the filter matches, the action you specify is invoked.

The filter is optional. If you do not specify a filter, the action is performed when any message is received. This is useful when debugging or when you want a catch-all message handler as the last message handler.

See “Using subject and content filters” on page 30 and “Using the filters message, message_start, and sender” on page 31.

**Action and altaction**

Each filter is associated with an action and, optionally, an alternative action called the altaction. If a message meets the conditions of the filter, the action is invoked. You must specify an action. If you specify an altaction, the altaction is invoked only if the action fails.

For each action and altaction, there can be one command, and it can be one of **start, run, post, socket, or DBLSN FULL SHUTDOWN**.

- **start** spawns a process. When you start a program, the Listener continues listening for more messages.

  When you start a program, the Listener doesn't wait for a return code, so it can only tell that the action has failed if it cannot find or start the program.

  The following example starts dbmlsync with some command line options, parts of which are obtained from the message using the $content action variable.

  "action='start dbmlsync.exe @dbmlsync.txt -n $content -wc dbmlsync_$content -e sch=INFINITE';"

- **run** runs the program and waits for it to finish. The Listener resumes listening after the process is complete.
When you run a program, the Listener determines that the program has failed if the Listener cannot find or start the program or if it returns a non-zero return code.

The following example runs dbmlsync with some command line options, parts of which are obtained from the message.

```
"action='run dbmlsync.exe @dbmlsync.txt -n $content';"
```

- **post** posts a Windows message to a window class. This is required by dbmlsync when scheduling is on. Post is also used when signaling applications that use Windows messages.

You can identify the Windows message by message contents or by the ID of the Windows message, if one exists.

You can identify the window class by its name or by the title of the window. If you identify the window class by name, you can use the dbmlsync -wc option to specify the window class name. If you identify the window class by its title, only the title of the top level window can be used to identify the window class.

If there are non-alphanumeric characters such as spaces or punctuation marks in your Windows message or window class name, you can put the message or name in single quotes. In that case, to use a single quote in the string, use two single quotes in a row. For example, to post my message to my class, use the following syntax:

```
... -l "action='post my''message to my''class';"
```

or

```
... -l "action='post ''my''''message'' to ''my''''class''';"
```

The following example posts a Windows message registered as dbas_synchronize to a dbmlsync instance registered with the class name dbmlsync_FullSync.

```
"action='post dbas_synchronize to dbmlsync_FullSync';"
```

See “-wc option” [MobiLink - Client Administration].

- **socket** notifies an application by making a TCP/IP connection. This is especially useful for passing dynamic information to a running application. It is also useful for integrating with Java and Visual Basic applications, because Java and VB don’t support custom window messaging, and eVB doesn’t support command line parameters. You can connect to a local socket by specifying just a port, or you can connect to a remote socket by specifying the host along with the port. Using sendText, you can send a string. You can optionally verify that the response is as expected with recvText. When you use recvText, you can also specify a timeout to avoid hanging if the case of application or network problems.

When you perform a socket action, the Listener determines that the action has failed if it failed to connect, send, or receive expected acknowledgement before timeout.

The following example forwards the string in $sender=$message to a local application that is listening on port 12345, and expects the application to send back "beeperAck" as an acknowledgement within 5 seconds.

```
-l "action='socket port=12345;
sendText=$sender=$message;""
recvText=beeperAck;
timeout=5'"

- **DBLSN FULL SHUTDOWN** causes the Listener utility to shut down. After shutdown, the Listener stops handling inbound messages and stops synchronizing device tracking information. The remote user must restart the Listener to continue with server-initiated synchronization. This feature is mostly useful during testing.

For example, action='DBLSN FULL SHUTDOWN'

You can only specify one action and one altaction in each instance of `-l`. If you want an action to perform multiple tasks, you can write a cover program or batch file that contains multiple actions, and run it as a single action.

Following is an example of altaction. In this example, $content is the protocol option for connecting to MobiLink. The primary action is to post the dbas_synchronize Windows message to the dbmlsync_FullSync window. The example uses altaction to start (not run) dbmlsync with the window class name dbmlsync_FullSync if the primary action fails. This is the standard way to make the Listener work with dbmlsync scheduling.

```
-l "subject=sync;
 action='post dbas_synchronize to dbmlsync_FullSync';
 altaction='start dbmlsync.exe
 @dbmlsync.txt
 -wc dbmlsync_FullSync
 -e adr=$content;sch=INFINITE''"
```

See also
- “Listeners” on page 28

### Action variables

The following Windows Listener action variables can be used anywhere in the action or altaction.

An action variable is substituted just before the action or altaction is performed.

Listener action variables start with a dollar sign ($). The escape character is also a dollar sign, so to specify a single dollar sign as plain text, type $$$. For example, type $$message_start when you don't want $message_start to be substituted.

<table>
<thead>
<tr>
<th>Action variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$subject</td>
<td>The subject of the message.</td>
</tr>
<tr>
<td>$content</td>
<td>The content of the message.</td>
</tr>
<tr>
<td>$message</td>
<td>The entire message, including subject, content, and formatting that is specific to the delivery path.</td>
</tr>
<tr>
<td>$message_start</td>
<td>A portion of the text of the message from the beginning, as specified in <code>-l message_start. This variable is only available if you have specified </code>-l message_start.</td>
</tr>
<tr>
<td>Action variable</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$message_end</td>
<td>The part of the message that is left over after the part specified in <code>-l message_start</code> is removed. This variable is only available if you have specified <code>-l message_start</code>.</td>
</tr>
<tr>
<td>$ml_connect</td>
<td>The MobiLink connection parameters specified by the mlsrv11 -x option. The default is an empty string.</td>
</tr>
<tr>
<td>$ml_user</td>
<td>The MobiLink user name as specified by dblsn -u, or the default name <em>(device-name-dblsn)</em>.</td>
</tr>
<tr>
<td>$ml_password</td>
<td>The MobiLink user name password as specified by dblsn -w, or the new MobiLink user name password if <code>-y</code> is used.</td>
</tr>
<tr>
<td>$priority</td>
<td>The meaning of this variable is carrier library-dependent.</td>
</tr>
<tr>
<td>$request_id</td>
<td>The request ID that was specified for the push request. See “Push requests” on page 10.</td>
</tr>
<tr>
<td>$remote_id</td>
<td>The remote ID. This variable can only be used when the dblsn -r option is specified. See “Filtering by remote ID” on page 30.</td>
</tr>
<tr>
<td>$sender</td>
<td>The sender of the message.</td>
</tr>
<tr>
<td>$type</td>
<td>The meaning of this variable is carrier library dependent.</td>
</tr>
<tr>
<td>$year</td>
<td>The meaning of this variable is carrier library-dependent.</td>
</tr>
<tr>
<td>$month</td>
<td>The meaning of this variable is carrier library-dependent. Values can be from 1-12.</td>
</tr>
<tr>
<td>$day</td>
<td>The meaning of this variable is carrier library-dependent. Values can be from 1-31.</td>
</tr>
<tr>
<td>$hour</td>
<td>The meaning of this variable is carrier library-dependent. Values can be from 0-23.</td>
</tr>
<tr>
<td>$minute</td>
<td>The meaning of this variable is carrier library-dependent. Values can be from 0-59.</td>
</tr>
<tr>
<td>$second</td>
<td>The meaning of this variable is carrier library-dependent. Values can be from 0-59.</td>
</tr>
<tr>
<td>$best_adapter_mac</td>
<td>The MAC address of the best NIC for reaching the MobiLink server that is specified in the dblsn command line with the -x option. If the best route does not go through a NIC, the value is an empty string.</td>
</tr>
<tr>
<td>Action variable</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$best_adapter_name</td>
<td>The adapter name of the best NIC for reaching the MobiLink server that is specified in the dblsn command line with the -x option. If the best route does not go through a NIC, the value is an empty string.</td>
</tr>
<tr>
<td>$best_ip</td>
<td>The IP address of the best IP interface for reaching the MobiLink server that is specified in the dblsn command line with the -x option. If that server is unreachable, the value is 0.0.0.0.</td>
</tr>
<tr>
<td>$best_network_name</td>
<td>The RAS or dialup profile name of the best profile for reaching the MobiLink server that is specified in the dblsn command line with the -x option. If the best route does not go through a RAS/dialup connection, the value is an empty string.</td>
</tr>
<tr>
<td>$adapters</td>
<td>A list of active network adapter names, each separated by a vertical bar (</td>
</tr>
<tr>
<td>$network_names</td>
<td>A list of connected RAS entry names, each separated by a vertical bar (</td>
</tr>
</tbody>
</table>

**Example**

For example, if a message arrives in the form `message_start pub-name`, you can use the following $message_end action variable to determine which publication to synchronize:

```
-l "message_start=message_start;action='dbmlsync.exe -c ... -n $message_end'"
```

**Listening libraries**

When you run the Windows Listener, by default the listening library `lsn_udp.dll` is used. If you are using SMTP, you need to specify an SMTP listening library.

You specify the listening library with the `dblsn -d` option, and specify options for the listening library with the `-a` option. To enable multi-channel listening, specify multiple DLLs by repeating `-d`. After each `-d` option, specify the `-a` and `-i` options that relate to the DLL. For example,

```
dblsn.exe -d lsn_udp.dll -i 10 -d maac750.dll -i 60
```

To specify multiple options, repeat `-a`. For example,

```
-d maac750.dll -a port=2439 -a ShowSenderPort
```

To see options for your DLL, type the following command:

```
dblsn -d filename.dll -a ?
```

Following is the supported listening library and its options.
UDP (lsn_udp.dll)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port=port-number</td>
<td>The default is 5001.</td>
</tr>
<tr>
<td>Timeout=seconds</td>
<td>This value must be smaller than the polling interval of the UDP listening</td>
</tr>
<tr>
<td></td>
<td>thread. The default is 0.</td>
</tr>
<tr>
<td>ShowSenderPort</td>
<td>All occurrences of the $sender action variable are followed by a :port-</td>
</tr>
<tr>
<td></td>
<td>number suffix.</td>
</tr>
<tr>
<td>HideWSAErrorBox</td>
<td>Suppresses the error box showing errors on socket operations.</td>
</tr>
<tr>
<td>CodePage=number</td>
<td>On Windows Mobile, translates multibyte characters into Unicode based</td>
</tr>
<tr>
<td></td>
<td>on this code page number.</td>
</tr>
</tbody>
</table>
CHAPTER 4

Listeners for Palm devices

Contents

Palm Listener utilities .................................................................................................................. 50
Palm Listener utilities

To run server-initiated synchronization on Palm devices, you use two utilities:

- Palm Listener Configuration utility (dblsncfg)
- Palm Listener (LsnT650.prc)

First, run the Palm Listener Configuration utility on a Windows desktop to create a configuration file for the Palm. The configuration file must later be transferred to the Palm device via HotSync.

For an overview of Listeners and message handlers, see “Listeners” on page 28.

Palm Listener Configuration utility

The Palm Listener Configuration utility, running on a Windows desktop, creates a configuration file for the Palm Listener. See “Palm Listener utility” on page 52.

Syntax

```
dblsncfg -n [ filename ] -l message-handler [ -l message-handler... ]
```

```
message-handler : [ filter; ] action
```

```
filter : [ subject = string ] [ content = string ] [ message = string | message_start = string ] [ sender = string ]
```

```
action : action=run application-name [ arguments ]
```

Parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>Reads options from the specified environment variable or configuration file. If both exist, the environment variable is used. See “Storing Listener options” on page 33.</td>
</tr>
<tr>
<td>-l message-handler</td>
<td>-l allows you to specify a message handler, which is a filter-action pair. The filter determines which message should be handled, and the action is invoked when the filter matches a message. You can specify multiple instances of -l. Each instance of -l specifies a different message handler.</td>
</tr>
<tr>
<td>-n filename</td>
<td>The -n option is used to create a configuration file for the Palm Listener. The filename should be LsnT650.pdb.</td>
</tr>
</tbody>
</table>
Filters  You specify a filter to compare to an incoming message. If the filter matches, the action you specify is invoked.

See “Using subject and content filters” on page 30 and “Using the filters message, message_start, and sender” on page 31.

The filter is optional. If you do not specify a filter, the action is performed when any message is received.

Action
The action fully launches the specified application. The syntax is run application-name [ arguments ].

`arguments` is an application-dependent string; it may contain action variables. The PilotMain routine of the target application should take a string as the command block. See “Palm action variables” on page 51.

Note: When running the Palm Listener Configuration utility on a Windows desktop to generate a configuration file for the Palm, you must specify the run action. However, on the Palm device you can delete the run action using the Handler Editor in the Palm Listener. This way you can consume the message without causing an action.

**Palm action variables**

The following Palm action variables can be used in the arguments in the run clause.

An action variable is substituted just before the action is performed.

Listener action variables start with a dollar sign ($). The escape character is also a dollar sign, so to specify a dollar sign as plain text, enter $$$. For example, enter $$message_start when you don't want $message_start to be substituted.

<table>
<thead>
<tr>
<th>Action variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$subject</td>
<td>The subject of the message.</td>
</tr>
<tr>
<td>$object</td>
<td>The object of the message.</td>
</tr>
<tr>
<td>$message</td>
<td>The full message string.</td>
</tr>
<tr>
<td>$message_start</td>
<td>A portion of the text of the message from the beginning, as specified in -l message_start. This variable is only available if you have specified -l message_start.</td>
</tr>
<tr>
<td>$message_end</td>
<td>The part of the message that is left over after the part specified in -l message_start is removed. This variable is only available if you have specified -l message_start.</td>
</tr>
<tr>
<td>$sender</td>
<td>The sender of the message.</td>
</tr>
<tr>
<td>$time</td>
<td>This is the current time in seconds since 12:00 AM, January 1, 1904.</td>
</tr>
</tbody>
</table>
Palm Listener utility

For Palm applications using server-initiated synchronization, each client must have a Palm Listener installed. The Listener files are:

- **LsnT650.prc** the Listener on Treo 650

Currently, the Palm Listeners only read from configuration file *lsncfg.pdb*.

The Palm Listener also allows you to set three options. These options remain until they are explicitly changed or until you perform a reset.

- **Listening** A way to stop the Listener from consuming messages.
- **Enable Actions** This is applicable only when Listening is on. When disabled, no action is invoked.
- **Prompt Before Actions** This is applicable only when actions are enabled. When this option is set, a confirmation window pops up before an action is invoked.

The device need not always be on if it turns on automatically when an SMS message is received. Treo devices do not need to be on for the Listener to work.

A Listener SDK is provided that you can use to create support for other Palm devices. See “MobiLink Listener SDK for Palm” on page 89.
CHAPTER 5

MobiLink Notifier properties

Contents

Common properties .................................................................................................. 54
Notifier properties ................................................................................................. 55
Gateway properties ................................................................................................. 66
Carrier properties .................................................................................................. 74
Common properties

There is one common property, verbosity.

For more information about setting properties, see “Setting properties” on page 14.

verbosity property

The verbosity setting applies to all Notifiers, gateways, and carriers. You can set the verbosity to the following levels:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No trace (the default)</td>
</tr>
<tr>
<td>1</td>
<td>Startup, shutdown, and property trace</td>
</tr>
<tr>
<td>2</td>
<td>Display notification messages</td>
</tr>
<tr>
<td>3</td>
<td>Full-level trace</td>
</tr>
</tbody>
</table>

See also

- “Setting properties” on page 14
Notifier properties

Notifier properties can be set in the Notifier properties file or stored in the MobiLink system tables. The enable and request_cursor properties are required. All other Notifier properties are optional.

There are three types of Notifier property:

Notifier events

The confirmation_handler event occurs asynchronously from the other Notifier events. All of these events are optional.

- “begin_connection property” on page 56
- “confirmation_handler property” on page 56
- “end_connection property” on page 59

Notifier polling events

When the Notifier polls the consolidated database, it invokes the polling events in the following order. All of these events are optional except for the request_cursor.

For each poll (begin_poll
  shutdown_query
  request_cursor
    For all requests expired before required confirmation (error_handler
    )
  request_delete
  end_poll
)

- “begin_poll property” on page 59
- “end_poll property” on page 60
- “error_handler property” on page 60
- “request_cursor property” on page 62
- “request_delete property” on page 62
- “shutdown_query property” on page 63

Notifier behavior properties

You can also configure the Notifier to use a variety of settings. All of these properties are optional.

- “connect_string property” on page 63
- “enable property” on page 63
- “gui property” on page 64
- “isolation property” on page 64
- “poll_every property” on page 64

See also

- “Notifiers” on page 17
- “Setting properties” on page 14
Notifier events

begin_connection property

This is a SQL statement that runs in a separate transaction after the Notifier connects to the database and before the first poll. For example, this property can be used to create temporary tables or variables.

If the Notifier loses its connection to the consolidated database, it re-executes this transaction immediately after reconnecting.

You should not use this property to change isolation levels. To control isolation levels, use the isolation property.

See also
● “Setting properties” on page 14
● “isolation property” on page 64

confirmation_handler property

You can implement this property to programmatically handle delivery confirmation information uploaded by remote listeners. If the status parameter is 0, the push request identified by request_id was successfully received by the Listener identified by the remote_device parameters.

You can use the request_option out parameter to take an appropriate action in response to the delivery confirmation. If request_option is 0, the confirmation_handler takes the default Notifier action: the request_delete event is executed to delete the original push request. However, if the Listener device sending the delivery confirmation does not match the Listener device identified by the request_id, the default action is to send the original push request on a secondary gateway.

Note: to enable remote Listeners to upload delivery confirmation information use the dblnsn -x option. If you want delivery confirmation but do not want IP tracking, use the dblnsn -ni option.

See -x and -ni in “Listener syntax” on page 38.

Following are the confirmation_handler parameters. You can use all of the parameters or a subset. This property requires the use of a stored procedure.
<table>
<thead>
<tr>
<th>Script parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_option (out)</td>
<td>Integer. Controls what the Notifier does to the request after the handler returns. Can be one of:</td>
</tr>
<tr>
<td></td>
<td>● 0: Perform default Notifier action based on the value of the status parameter. If status indicates that the responding device is the target one, then the Notifier deletes the request; otherwise the Notifier attempts to deliver on a secondary gateway.</td>
</tr>
<tr>
<td></td>
<td>● 1: Do nothing.</td>
</tr>
<tr>
<td></td>
<td>● 2: Execute Notifier.request_delete.</td>
</tr>
<tr>
<td></td>
<td>● 3: Attempt to deliver to a secondary gateway.</td>
</tr>
<tr>
<td>status (in)</td>
<td>Integer. Summary of the situation. The status can be used during development to catch problems such as incorrect filters and handler attributes. The status can be one of:</td>
</tr>
<tr>
<td></td>
<td>● 0: Received and confirmed.</td>
</tr>
<tr>
<td></td>
<td>● -2: Right respondent but the message was rejected.</td>
</tr>
<tr>
<td></td>
<td>● -3: Right respondent and the message was accepted but the action failed.</td>
</tr>
<tr>
<td></td>
<td>● -4: Wrong respondent and the message was accepted.</td>
</tr>
<tr>
<td></td>
<td>● -5: Wrong respondent and the message was rejected.</td>
</tr>
<tr>
<td></td>
<td>● -6: Wrong respondent. The message was accepted and the action succeeded.</td>
</tr>
<tr>
<td></td>
<td>● -7: Wrong respondent. The message was accepted but the action failed.</td>
</tr>
<tr>
<td>request_id (in)</td>
<td>Integer. Identifies the request.</td>
</tr>
<tr>
<td>remote_code (in)</td>
<td>Integer. Summary reported by the remote Listener. Can be one of:</td>
</tr>
<tr>
<td></td>
<td>● 1: Message accepted.</td>
</tr>
<tr>
<td></td>
<td>● 2: Message rejected.</td>
</tr>
<tr>
<td></td>
<td>● 3: Message accepted and action succeeded.</td>
</tr>
<tr>
<td></td>
<td>● 4: Message accepted and action failed.</td>
</tr>
<tr>
<td>remote_device (in)</td>
<td>Varchar. Device name of the responding Listener.</td>
</tr>
<tr>
<td>remote_mluser (in)</td>
<td>Varchar. MobiLink user name of the responding Listener.</td>
</tr>
<tr>
<td>remote_action_return (in)</td>
<td>Varchar. Return code of the remote action.</td>
</tr>
<tr>
<td>remote_action (in)</td>
<td>Varchar. Reserved for the action command.</td>
</tr>
<tr>
<td>gateway (in)</td>
<td>Varchar. Gateway associated with the request.</td>
</tr>
<tr>
<td>address (in)</td>
<td>Varchar. Address associated with the request.</td>
</tr>
<tr>
<td>subject (in)</td>
<td>Varchar. Subject associated with the request.</td>
</tr>
<tr>
<td>Script parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>content (in)</td>
<td>Varchar. Content associated with the request.</td>
</tr>
</tbody>
</table>

See also

- Device tracker gateways: “confirm_delivery property” on page 66
- SMTP gateways: “confirm_delivery property” on page 68
- UDP gateways: “confirm_delivery property” on page 72
- “ml_add_property system procedure” [MobiLink - Server Administration]

Example

In the following example, you create a table called CustomConfirmation and then you log confirmations to it using a stored procedure called CustomConfirmationHandler. In this example, the output parameter request_option is always set to 0, which means that default Notifier handling is used.

```sql
CREATE TABLE CustomConfirmation(
  error_code   integer,
  request_id   integer,
  remote_code   integer,
  remote_device   varchar(128),
  remote_mluser   varchar(128),
  remote_action_return   varchar(128),
  remote_action   varchar(128),
  gateway   varchar(255),
  address   varchar(255),
  subject   varchar(255),
  content   varchar(255),
  occurAt   timestamp not null default timestamp )

CREATE PROCEDURE CustomConfirmationHandler(
  out @request_option integer,
  in @error_code   integer,
  in @request_id   integer,
  in @remote_code   integer,
  in @remote_device   varchar(128),
  in @remote_mluser   varchar(128),
  in @remote_action_return   varchar(128),
  in @remote_action   varchar(128),
  in @gateway   varchar(255),
  in @address   varchar(255),
  in @subject   varchar(255),
  in @content   varchar(255) )

begin
  INSERT INTO CustomConfirmation(
    error_code, request_id, remote_code, remote_device, remote_mluser, remote_action_return, remote_action, gateway, address, subject, content )
  VALUES (@error_code, @request_id,)
```

Copyright © 2008, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.0
@remote_code,
@remote_device,
@remote_mluser,
@remote_action_return,
@remote_action,
@gateway,
@address,
@subject,
@content );
    SET @request_option = 0;
end

end_connection property

This is a SQL statement that runs as a separate transaction just before a Notifier database connection is closed. For example, this property can be used to delete temporary storage such as SQL variables and temporary tables.

The statement is executed in a standalone transaction.

Notifier polling events

begin_poll property

This is a SQL statement that is executed before each Notifier poll. Typical uses are to detect data change in the database and create push requests that are later fetched with the request_cursor.

The statement is executed in a standalone transaction.

This property is optional. The default is null.

See also
- “Setting properties” on page 14

Example

This example creates a push request for a Notifier called Notifier A. It uses a SQL statement that inserts rows into a table called PushRequest. Each row in this table represents a message to send to an address. The WHERE clause determines what push requests are inserted into the PushRequest table.

To use the stored procedure ml_add_property with a SQL Anywhere consolidated database, run the following command:

```sql
ml_add_property( 'SIS',
    'Notifier(Notifier A)',
    'begin_connection',
    'INSERT INTO PushRequest
    ( gateway, mluser, subject, content )
    SELECT ''MyGateway'', DISTINCT mluser,
    ''sync'', stream_param
    FROM MLUserExtra, mluser_union, Dealer
```
WHERE
MLUserExtra.mluser = mluser_union.name
AND( push_sync_status = 'waiting for request'
  OR datediff( hour, last_status_change, now() ) > 12 )
AND ( mluser_union.publication_name is NULL
  OR mluser_union.publication_name = 'FullSync' )
AND
  Dealer.last_modified > mluser_union.last_sync_time'
);

end_poll property

This is a SQL statement that is executed after each poll. Typical uses are to perform customized cleanup or track polling.

The statement is executed in a standalone transaction.

This property is optional. The default is null.

error_handler property

You can implement this property to catch situations where a transmission failed or was not confirmed. For example, when a transmission fails you can cause a line to be inserted in an audit table or a notification sent to someone.

You can capture the following information. You can use all of the parameters or a subset. This property requires the use of a stored procedure.

<table>
<thead>
<tr>
<th>Script parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request_option</td>
<td>Integer. Controls what the Notifier does to the request after the handler returns. Can be one of:</td>
</tr>
<tr>
<td>(out)</td>
<td>● 0: Perform default action based on the error code and log the error.</td>
</tr>
<tr>
<td></td>
<td>● 1: Do nothing.</td>
</tr>
<tr>
<td></td>
<td>● 2: Execute Notifier.request_delete.</td>
</tr>
<tr>
<td></td>
<td>● 3: Attempt to deliver to a secondary gateway.</td>
</tr>
<tr>
<td>error_code (in)</td>
<td>Integer. Can be one of:</td>
</tr>
<tr>
<td></td>
<td>● -1: The request timed out with confirmation of success.</td>
</tr>
<tr>
<td></td>
<td>● -8: Error during delivery attempt.</td>
</tr>
<tr>
<td>request_id (in)</td>
<td>Integer. Identifies the request.</td>
</tr>
<tr>
<td>gateway (in)</td>
<td>Varchar. Gateway associated with the request.</td>
</tr>
<tr>
<td>address (in)</td>
<td>Varchar. Address associated with the request.</td>
</tr>
</tbody>
</table>
### Script parameter | Description
--- | ---
subject (in) | Varchar. Subject associated with the request.
content (in) | Varchar. Content associated with the request.

#### See also
- “ml_add_property system procedure” [MobiLink - Server Administration]

#### Example

In the following example, you create a table called `CustomError`. You log errors to the table using a stored procedure called `CustomErrorHandler`. In this example, the output parameter `notifier_opcode` is always 0, which means that default Notifier handling is used.

```sql
CREATE TABLE CustomError(
    error_code  integer,
    request_id  integer,
    gateway     varchar(255),
    address     varchar(255),
    subject     varchar(255),
    content     varchar(255),
    occurAt     timestamp not null default timestamp
);

CREATE PROCEDURE CustomErrorHandler(
    out @notifier_opcode integer,
    in  @error_code   integer,
    in  @request_id   integer,
    in  @gateway      varchar(255),
    in  @address      varchar(255),
    in  @subject      varchar(255),
    in  @content      varchar(255) )
begin
    INSERT INTO CustomError(
        error_code,
        request_id,
        gateway,
        address,
        subject,
        content )
    VALUES(
        @error_code,
        @request_id,
        @gateway,
        @address,
        @subject,
        @content )
    set @notifier_opcode = 0;
end
```

To use the stored procedure `ml_add_property` with a SQL Anywhere consolidated database, run the following command:

```sql
call ml_add_property(
    'SIS',
    'Notifier(myNotifier)',
    'error_handler',
    'call CustomErrorHandler(?, ?, ?, ?, ?, ?, ?)');
```
**request_cursor property**

This property contains SQL used by the Notifier to fetch push requests. Each row is a push request that determines what information is sent in the message, who receives the information, when, and where. You must set this property.

The result set of this statement must contain at least five columns, and can optionally contain two other columns. These columns can have any name, but must be in the following order in the result set:

- request id
- gateway
- subject
- content
- address
- resend interval (optional)
- time to live (optional)

For more information about these columns, see “Creating the push request table” on page 10.

You might want to include a WHERE clause in your request_cursor to filter out requests that have been satisfied. For example, you can add a column to your push request table to track the time you inserted a request, and then use a WHERE clause to filter out requests that were inserted prior to the last time the user synchronized.

The statement is executed in a standalone transaction.

**request_delete property**

This is a SQL statement that specifies cleanup operations. The statement takes the request id as its only parameter. A parameter can be referenced by a named parameter or using a question mark (?).

Using the DELETE statement, the Notifier can automatically remove these forms of old request:

- **implicitly dropped requests** requests that appeared previously, but did not appear in the current set of requests obtained from the request_cursor.
- **confirmed requests** messages confirmed as delivered.
- **expired requests** requests that have expired based on their resend attributes and the current time. Requests without resend attributes are considered expired even if they appear in the next request.

The request_delete statement is executed per request ID in a standalone transaction when the need for deletion is detected. It is optional if you have provided another process to do the cleanup.

You can write the request_delete script in such a way to avoid eliminating expired or implicitly dropped requests. For example, the CarDealer sample uses request_delete to set the status field of the PushRequest table to 'processed'.

```
UPDATE PushRequest SET status='processed' WHERE req_id = ?
```
The sample's begin_poll script uses the last synchronization time to check that a remote device is up-to-date prior to eliminating processed requests.

For more information, see the Car Dealer sample located in samples-dir\MobiLink\SIS_CarDealer. (For information about samples-dir, see “Samples directory” [SQL Anywhere Server - Database Administration].)

**shutdown_query property**

This is a SQL statement that is executed right after begin_poll. The result should contain only the value yes (or 1) or no (or 0). To shut down the Notifier, specify yes or 1. This statement is executed as a standalone transaction.

If you are storing the shutdown state in a table, then you can use the end_connection property to reset the state before the Notifier disconnects.

**Notifier behavior properties**

**connect_string property**

By default, the Notifier uses ianywhere.ml.script.ServerContext to connect to the consolidated database. This means that it uses the connection string that was specified in the current mlsrv11 session's command line.

This is an optional property that can be used to override the default connection behavior. You can use it to connect to any database, including the consolidated database. It may be useful to connect to another database when you want notification logic and data to be separate from your synchronization data. Most deployments do not set this property.

See “ServerContext interface” [MobiLink - Server Administration].

See also

- “Setting properties” on page 14

**enable property**

You can enable or disable existing Notifiers. If you have enabled multiple Notifiers, all are started when you start the MobiLink server with the -notifier option.

See also

- “Setting properties” on page 14
gui property

This controls whether the Notifier window is displayed on the computer where the Notifier is running. This user interface allows users to temporarily change the polling interval, or to poll immediately. It can also be used to shut down the Notifier without shutting down the MobiLink server. (Once stopped, the Notifier can only be restarted by shutting down and restarting the MobiLink server.)

This property is optional. The default is ON.

See also

● “Setting properties” on page 14

isolation property

Isolation is an optional property that controls the isolation level of the Notifier's database connection. The default value is 1. You can use the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Isolation level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Read uncommitted</td>
</tr>
<tr>
<td>1</td>
<td>Read committed (the default)</td>
</tr>
<tr>
<td>2</td>
<td>Repeatable read</td>
</tr>
<tr>
<td>3</td>
<td>Serializable</td>
</tr>
</tbody>
</table>

Remarks

Be aware of the consequences of setting the isolation level. Higher levels increase contention, and may adversely affect performance. Isolation level 0 allows reads of uncommitted data—data which may eventually be rolled back.

See also

● “Setting properties” on page 14

poll_every property

This property specifies the Notifier polling interval. You can specify S, M, and H for units of seconds, minutes, and hours. You can also combine units, as in 1H 30M 10S. If no unit is specified, the interval is in seconds.

If the Notifier loses the database connection, it recovers automatically at the first polling interval after the database becomes available again.

This property is optional. The default is 30 seconds.
shared_database_connection property

Set this option to on if you want Notifiers to share connections.

When Notifiers share connections they consume fewer system resources without incurring performance penalties. However, in some situations it is not possible to share connections, such as when applications use SQL variables that have non-unique names among Notifiers.

Notifiers can share connections only if they are at the same isolation level.

The default is off.

See also

- “isolation property” on page 64
Gateway properties

Device tracker gateway properties

To use the default device tracker gateway, include the name **Default-DeviceTracker** in the second column of the result set of the request_cursor.

A device tracker gateway utilizes automatically-tracked IP addresses, phone numbers, and public wireless network provider IDs to deliver messages through either a UDP or SMTP gateway. Your configuration defines which UDP gateway and which SMTP gateway are to be used by your device tracker gateway. You can also control tracking requirements for messages sent through this gateway.

**confirm_delivery property**

Specifies whether the Listener should confirm with the consolidated database that the message was received. To be able to do this, you must start the Listener with the -x option. The default setting for confirm_delivery is yes.

For information about handling delivery confirmation on the server, see “confirmation_handler property” on page 56.

For example,

```
DeviceTracker(Default-DeviceTracker).confirm_delivery = yes
```

See also

- “Device tracking” on page 22
- “Setting properties” on page 14

**enable property**

Specify **enable=yes** to use a device tracker gateway. Specify **enable=no** to disable a device tracker gateway. You can define and use multiple device tracker gateways.

See also

- “Device tracking” on page 22
- “Setting properties” on page 14

**smtp_gateway property**

This names an SMTP gateway that the device tracker can use. The gateway must be enabled. A device tracker gateway can only use one SMTP gateway. The default is Default-SMTP.
See also

- “SMTP gateway properties” on page 67
- “Device tracking” on page 22
- “Setting properties” on page 14

**sync_gateway property**

This names a SYNC gateway that the device tracker can use. The gateway must be enabled. A device tracker gateway can only use one SYNC gateway.

The SYNC gateway is a TCP/IP-based gateway that supports notification through a persistent connection. It is the recommended gateway.

There is a predefined instance of the SYNC gateway called Default-SYNC. It is configured to work with the predefined device tracker gateway, called Default-DeviceTracker. When the Default-SYNC gateway is used through the Default-DeviceTracker gateway, delivery confirmation is turned on by default.

See also

- “Device tracking” on page 22
- “Setting properties” on page 14

**udp_gateway property**

This identifies a UDP gateway that the device tracker can use. The gateway must be enabled. A device tracker gateway can only use one UDP gateway. The default is Default-UDP.

See also

- “UDP gateway properties” on page 71
- “Device tracking” on page 22
- “Setting properties” on page 14

**SMTP gateway properties**

SMTP gateway configuration is required only if you need to send SMS messages via SMTP.

SMTP gateways can be used to send email messages. In particular, they can send SMS messages to SMS listeners via a wireless carrier's email-to-SMS service.

In the following list of properties, the enable and server properties are required. The server and sender properties are often required. The user and password properties may be required, depending on your SMTP server setup. All other SMTP gateway properties are optional.

You can have multiple SMTP gateways. To set up additional SMTP gateways, copy the properties for one gateway and provide a different gateway name and property values.

For more information about gateways, see “Gateways and carriers” on page 20.
**confirm_delivery property**

Specify yes to confirm delivery. The default is no. This property has an effect only when sending directly through this gateway (not indirectly via a device tracking gateway).

**See also**
- “confirmation_handler property” on page 56
- “Device tracking” on page 22
- “Setting properties” on page 14

**confirm_timeout property**

Specify the amount of time before a confirmation should time out. Specify s, m, or h for seconds, minutes, or hours. If you do not specify s, m, or h, the default is seconds.

The default confirmation timeout is 10m.

**See also**
- “Device tracking” on page 22
- “Setting properties” on page 14

**description property**

Use this property to optionally describe the gateway.

**See also**
- “Device tracking” on page 22
- “Setting properties” on page 14

**enable property**

Specify enable=yes to use an SMTP gateway. You can define and use multiple SMTP gateways.

**See also**
- “Device tracking” on page 22
- “Setting properties” on page 14

**listeners_are_900 property**

Specify yes if all Listeners are Adaptive Server Anywhere version 9.0.0 clients. Specify no if they are version 9.0.1 or later. The default is no.
password property

This is the password for your SMTP service. Your SMTP service may not require a password.

See also

● “Device tracking” on page 22
● “Setting properties” on page 14

sender property

This is the sender address of the emails (SMTP requests). The default is anonymous.

The sender may not be available as an action variable to the Listener if the arriving message format is not compatible with the MobiLink message interpretation.

See also

● “Device tracking” on page 22
● “Setting properties” on page 14

server property

This is the IP address or host name of the SMTP server used to send the message to the Listener. The default is mail.

See also

● “Device tracking” on page 22
● “Setting properties” on page 14

user property

This is the user name for your SMTP service. Your SMTP service may not require a user name.

See also

● “Device tracking” on page 22
● “Setting properties” on page 14
SYNC gateway properties

SYNC gateway configuration is required only if you need to send notification over the same protocol as your synchronizations.

A default SYNC gateway is provided and enabled by default. This default gateway can be used only if Notifier configuration is stored in the consolidated database (and not in a Notifier properties file).

The SYNC gateway uses persistent connections. The same connection is used for notifications, confirmations, and device tracking. The SYNC gateway is the preferred gateway: when using device tracking, a notification attempt starts with the SYNC gateway, and fallback to the UDP gateway and then the SMTP gateway.

In the following list of properties, only the enable property is required. All other SYNC gateway properties are optional.

You can have multiple SYNC gateways. To set up additional SYNC gateways, you can copy the properties for one gateway and provide a different gateway name and property values.

For more information about gateways, see “Gateways and carriers” on page 20.

For more information about setting properties, see “Setting properties” on page 14.

confirm_action property

Specify yes if you want confirmation upon delivery. This property has an effect only when sending directly through this gateway, and not when sending indirectly through a device tracking gateway.

The default is no.

See also

- “confirmation_handler property” on page 56
- “Setting properties” on page 14

confirm_delivery property

Specify yes to confirm delivery. The default is no. This property has an effect only when sending directly through this gateway (not indirectly via a device tracking gateway).

See also

- “confirmation_handler property” on page 56
- “Setting properties” on page 14
**confirm_timeout property**

Specify the amount of time before a confirmation should time out. Specify s, m, or h for seconds, minutes, or hours. If you do not specify s, m, or h, the default is seconds.

The default confirmation timeout is 1m.

**See also**
- “Setting properties” on page 14

**description property**

Use this property to optionally describe the gateway.

**See also**
- “Setting properties” on page 14

**enable property**

Specify `enable=yes` to use a SYNC gateway.

**See also**
- “Setting properties” on page 14

**UDP gateway properties**

UDP gateway configuration is required only if you need to send UDP messages.

The format of the UDP message is `[subject] content`, where `subject` and `content` come from the subject and content columns of the request_cursor Notifier property.

In the following list of properties, only the enable property is required. All other UDP gateway properties are optional.

You can have multiple UDP gateways. To set up additional UDP gateways, copy the properties for one gateway and provide a different gateway name and property values.

For more information about gateways, see “Gateways and carriers” on page 20.

For more information about setting properties, see “Setting properties” on page 14.
**confirm_delivery property**

Specify yes to confirm delivery. The default is yes. This property has an effect only when sending directly through this gateway (not indirectly via a device tracking gateway).

**See also**
- “confirmation_handler property” on page 56
- “Setting properties” on page 14

**confirm_timeout property**

Specify the amount of time before a confirmation should time out. Specify s, m, or h for seconds, minutes, or hours. If you do not specify s, m, or h, the default is seconds.

The default confirmation timeout is 1m.

**See also**
- “Setting properties” on page 14

**description property**

Use this property to optionally describe the gateway.

**See also**
- “Setting properties” on page 14

**enable property**

Specify `enable=yes` to use a UDP gateway. You can define and use multiple UDP gateways.

**See also**
- “Setting properties” on page 14

**listeners_are_900 property**

Specify yes if all Listeners are Adaptive Server Anywhere version 9.0.0 clients. Specify no if they are version 9.0.1 or later. The default is no.

**See also**
- “Setting properties” on page 14
**listener_port property**

This is the port on the remote device where the gateway sends the UDP packet. This property is optional. The default is the default listening port of the UDP Listener (5001).

See also
- “Setting properties” on page 14

**sender property**

This is the IP address or host name of the sender. This property is optional, and is only useful for multi-homed hosts. The default is localhost.

See also
- “Setting properties” on page 14

**sender_port property**

This is the port to use for sending the UDP packet. This property is optional; you may need to set it if your firewall restricts outgoing traffic. If not set, your operating system assigns a free port.

See also
- “Setting properties” on page 14
Carrier properties

Carriers are required only when you are using an SMTP gateway.

Carrier properties set up public wireless carrier configuration, which provides carrier-specific information such as how to map automatically-tracked phone numbers and network providers to SMS email addresses.

Carrier information is used when the device tracker gateway needs an SMS email address to be generated from an automatically-tracked device address. Addresses are generated in the following form:

\[
\text{email-address} = \text{sms_email_user_prefix}\text{phone-number}@\text{sms_email_domain}
\]

where:

- \text{sms_email_user_prefix} is the value of the \text{sms_email_user_prefix} property
- the phone number comes from the \text{ml_device_address.address} column
- \text{sms_email_domain} is the value of the \text{sms_email_domain} property

See also

- “\text{sms_email_domain} property” on page 75
- “\text{sms_email_user_prefix} property” on page 75
- “\text{ml_device_address}” [MobiLink - Server Administration]

For more information about carriers, see “Gateways and carriers” on page 20.

For more information about setting properties, see “Setting properties” on page 14.

enable property

Specify \text{enable=\text{yes}} to use a Carrier mapping. You can define and use multiple Carrier mappings in one file.

See also

- “Setting properties” on page 14

network_provider_id property

Specifies the network provider ID.

To use SMS on CE Phone Edition, set the network provider ID to _generic_. For example,

\[
\text{network_provider_id=\_generic\_}
\]

See also

- “Setting properties” on page 14
sms_email_domain property

Specifies the domain name of the carrier.

Carrier information is used when the device tracker gateway needs an SMS email address to be generated from an automatically-tracked device address. Addresses are generated in the following form:

\[
\text{email-address} = \text{sms_email_user_prefix}\text{phone-number}@\text{sms_email_domain}
\]

where:

● \text{sms_email_user_prefix} is the value of the sms_email_user_prefix property
● the phone number comes from the ml_device_address.address column
● \text{sms_email_domain} is the value of the sms_email_domain property

See also

● “sms_email_user_prefix property” on page 75
● “ml_device_address” [MobiLink - Server Administration]
● “Setting properties” on page 14

sms_email_user_prefix property

Specifies the prefix used in email addresses.

Carrier information is used when the device tracker gateway needs an SMS email address to be generated from an automatically-tracked device address. Addresses are generated in the following form:

\[
\text{email-address} = \text{sms_email_user_prefix}\text{phone-number}@\text{sms_email_domain}
\]

where:

● \text{sms_email_user_prefix} is the value of the sms_email_user_prefix property
● the phone number comes from the ml_device_address.address column
● \text{sms_email_domain} is the value of the sms_email_domain property

See also

● “sms_email_domain property” on page 75
● “ml_device_address” [MobiLink - Server Administration]
● “Setting properties” on page 14
CHAPTER 6

Server-initiated synchronization system procedures

Contents

Introduction to server-initiated synchronization system procedures .................. 78
IBM DB2 mainframe server-initiated synchronization system procedure name conversions .......................................................................................................................... 79
ml_delete_device ....................................................................................................... 80
ml_delete_device_address ....................................................................................... 81
ml_delete_listening ................................................................................................... 82
ml_set_device ............................................................................................................ 83
ml_set_device_address ............................................................................................ 85
ml_set_listening ........................................................................................................ 87
Introduction to server-initiated synchronization system procedures

Server-initiated synchronization system procedures add and delete rows in MobiLink system tables.

**Note**
These system procedures are used for device tracking. If you use remote devices that support automatic device tracking, you do not need to use these system procedures. If you use remote devices that do not support automatic device tracking, you can configure manual device tracking using these system procedures.

For more information, see “Device tracking” on page 22 and “Using device tracking with Listeners that don't support it” on page 24.

For more information about MobiLink system tables, see “MobiLink server system tables” [MobiLink - Server Administration].

For information about other MobiLink system procedures, see “MobiLink server system procedures” [MobiLink - Server Administration].
IBM DB2 mainframe server-initiated synchronization system procedure name conversions

DB2 mainframe version 8.1 supports a backward compatibility mode, where column names and other identifiers are limited to a maximum of 18 characters. To support this environment, all MobiLink system objects in DB2 mainframe have names of 18 characters or less. The following table identifies how procedure names for DB2 mainframe consolidated databases are mapped to system procedure names for all other consolidated database types.

If the system procedure name does not appear in the table below, no conversion is required.

<table>
<thead>
<tr>
<th>System procedure name</th>
<th>System procedure name for DB2 mainframe consolidated databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>“ml_delete_device_address” on page 81</td>
<td>ml_del_dev_addr</td>
</tr>
<tr>
<td>“ml_delete_listening” on page 82</td>
<td>ml_del_listen</td>
</tr>
<tr>
<td>“ml_set_device_address” on page 85</td>
<td>ml_set_dev_addr</td>
</tr>
</tbody>
</table>
ml_delete_device

Use this system procedure to delete all information about a remote device when you are manually setting up device tracking.

Parameters

<table>
<thead>
<tr>
<th>Item</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>device</td>
<td>VARCHAR(255). Device name.</td>
</tr>
</tbody>
</table>

Remarks

This function is useful only if you are manually setting up device tracking.

See “Using device tracking with Listeners that don't support it” on page 24.

Example

Delete a device record and all associated records that reference this device record:

    CALL ml_delete_device( 'myOldDevice' );
**ml_delete_device_address**

Use this system procedure to delete a device address when you are manually setting up device tracking.

**Parameters**

<table>
<thead>
<tr>
<th>Item</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>device</td>
<td>VARCHAR(255)</td>
</tr>
<tr>
<td>2</td>
<td>medium</td>
<td>VARCHAR(255)</td>
</tr>
</tbody>
</table>

**Remarks**

This system procedure is useful only if you are manually setting up device tracking.

For DB2 mainframe consolidated database types, this procedure is called `ml_del_dev_addr`. See “IBM DB2 mainframe server-initiated synchronization system procedure name conversions” on page 79.

See “Using device tracking with Listeners that don't support it” on page 24.

**Example**

Delete an address record:

```sql
CALL ml_delete_device_address( 'myFirstTreo180', 'ROGERS AT&T' );
```
ml_delete_listening

Use this system procedure to delete mappings between a MobiLink user and remote devices when you are manually setting up device tracking.

Parameters

<table>
<thead>
<tr>
<th>Item</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ml_user</td>
<td>VARCHAR(128)</td>
</tr>
</tbody>
</table>

Remarks

This system procedure is useful only if you are manually setting up device tracking.

For DB2 mainframe consolidated database types, this procedure is called ml_del_listen. See “IBM DB2 mainframe server-initiated synchronization system procedure name conversions” on page 79.

See “Using device tracking with Listeners that don't support it” on page 24.

Example

Delete a recipient record:

```sql
CALL ml_delete_listening( 'myULDB' );
```
ml_set_device

Use this system procedure to add or alter information about remote devices when you are manually setting up device tracking. It adds or updates a row in the ml_device table.

**Parameters**

<table>
<thead>
<tr>
<th>Item</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>device</td>
<td>VARCHAR(255). User-defined unique device name.</td>
</tr>
<tr>
<td>2</td>
<td>listener_version</td>
<td>VARCHAR(128). Optional remarks on listener version.</td>
</tr>
<tr>
<td>3</td>
<td>listener_protocol</td>
<td>INTEGER. Use 0 for version 9.0.0, 1 for post-9.0.0 Palm Listeners, 2 for post-9.0.0 Windows Listeners.</td>
</tr>
<tr>
<td>4</td>
<td>info</td>
<td>VARCHAR(255). Optional device information.</td>
</tr>
<tr>
<td>5</td>
<td>ignore_tracking</td>
<td>CHAR(1). Set to y to ignore tracking and stop it from overwriting manually entered data.</td>
</tr>
<tr>
<td>6</td>
<td>source</td>
<td>VARCHAR(255). Optional remarks on the source of this record.</td>
</tr>
</tbody>
</table>

**Remarks**

The system procedures ml_set_device, ml_set_device_address, and ml_set_listening are used to override automatic device tracking by changing information in the MobiLink system tables ml_device, ml_device_address, and ml_listening. For example, if some of your remote devices are Palm devices you may want to use automatic device tracking but manually insert data for the Palm devices.

This system procedure is useful only if you are manually setting up device tracking.

See “Using device tracking with Listeners that don't support it” on page 24.

**See also**

- “ml_set_device_address” on page 85
- “ml_set_listening” on page 87
- “ml_device” [MobiLink - Server Administration]
- “ml_device_address” [MobiLink - Server Administration]
- “ml_listening” [MobiLink - Server Administration]

**Example**

For each device, add a device record:

```sql
CALL ml_set_device(
  'myFirstTreo180',
  'MobiLink Listeners for Treo 180 - 9.0.1',
  '1',
  'not used',
);```
Server-initiated synchronization system procedures

'y',
'manually entered by administrator' );
ml_set_device_address

Use this system procedure to add or alter information about remote device addresses when you are manually setting up device tracking. It adds or updates a row in the ml_device_address table.

### Parameters

<table>
<thead>
<tr>
<th>Item</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>device</td>
<td>VARCHAR(255). Existing device name.</td>
</tr>
<tr>
<td>2</td>
<td>medium</td>
<td>VARCHAR(255). Network provider ID (must match a carrier's network_provider_id property).</td>
</tr>
<tr>
<td>3</td>
<td>address</td>
<td>VARCHAR(255). Phone number of an SMS-capable device.</td>
</tr>
<tr>
<td>4</td>
<td>active</td>
<td>CHAR(1). Set to y to activate this record to be used for sending notification.</td>
</tr>
<tr>
<td>5</td>
<td>ignore_tracking</td>
<td>CHAR(1). Set to y to ignore tracking and stop it from overwriting manually entered data.</td>
</tr>
<tr>
<td>6</td>
<td>source</td>
<td>VARCHAR(255). Optional remarks on the source of this record.</td>
</tr>
</tbody>
</table>

### Remarks

The system procedures ml_set_device, ml_set_device_address, and ml_set_listening are used to override automatic device tracking by changing information in the MobiLink system tables ml_device, ml_device_address, and ml_listening. For example, if some of your remote devices are Palms you may want to use automatic device tracking but manually insert data for the Palm devices.

This system procedure is useful only if you are manually setting up device tracking.

For DB2 mainframe consolidated database types, this procedure is called ml_set_dev_addr. See “IBM DB2 mainframe server-initiated synchronization system procedure name conversions” on page 79.

See “Using device tracking with Listeners that don't support it” on page 24.

### See also

- “ml_set_device” on page 83
- “ml_set_listening” on page 87
- “ml_device” [MobiLink - Server Administration]
- “ml_device_address” [MobiLink - Server Administration]
- “ml_listening” [MobiLink - Server Administration]

### Example

For each device, add an address record for a device:

```sql
ml_set_device_address
```
CALL ml_set_device_address(
    'myFirstTreo180',
    'ROGERS AT&T',
    '3211234567',
    'y',
    'y',
    'manually entered by administrator' );
ml_set_listening

Use this system procedure to add or alter mappings between MobiLink users and remote devices when you are manually setting up device tracking. It adds or updates a row in the ml_listening table.

### Parameters

<table>
<thead>
<tr>
<th>Item</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ml_user</td>
<td>VARCHAR(128). MobiLink user name.</td>
</tr>
<tr>
<td>2</td>
<td>device</td>
<td>VARCHAR(255). Existing device name.</td>
</tr>
<tr>
<td>3</td>
<td>listening</td>
<td>CHAR(1). Set to y to activate this record to be used for DeviceTracker addressing.</td>
</tr>
<tr>
<td>5</td>
<td>ignore_tracking</td>
<td>CHAR(1). Set to y to ignore tracking and stop it from over-writing manually entered data.</td>
</tr>
<tr>
<td>6</td>
<td>source</td>
<td>VARCHAR(255). Optional remarks on the source of this record.</td>
</tr>
</tbody>
</table>

### Remarks

The system procedures ml_set_device, ml_set_device_address, and ml_set_listening are used to override automatic device tracking by changing information in the MobiLink system tables ml_device, ml_device_address, and ml_listening. For example, if some of your remote devices are Palms you may want to use automatic device tracking but manually insert data for the Palm devices.

This system procedure is useful only if you are manually setting up device tracking.

See “Using device tracking with Listeners that don't support it” on page 24.

### See also

- “ml_set_device” on page 83
- “ml_set_device_address” on page 85
- “ml_device” [MobiLink - Server Administration]
- “ml_device_address” [MobiLink - Server Administration]
- “ml_listening” [MobiLink - Server Administration]

### Example

For each remote database, add a recipient record for a device. This maps the device to the MobiLink user name.

```sql
CALL ml_set_listening(
    'myULDB',
    'myFirstTreo180',
    'y',
    'y',
    'manually entered by administrator' );
```
CHAPTER 7

MobiLink Listener SDK for Palm

Contents

Introduction to MobiLink Listener SDK for Palm ................................................... 90
Message processing interface ................................................................................. 91
Device dependent functions ................................................................................... 100
Introduction to MobiLink Listener SDK for Palm

You can use the Listener SDK for Palm to create Listeners for new Palm devices. The Listener SDK is a simple API that is provided to help you extend the Listener utility. The programming interface includes a message processing interface and device dependent functions. You can use the Listener SDK to create Listeners for new Palm devices or new wireless network adapters.

Palm Listener SDK files

The MobiLink Listener SDK and sample implementations are located in the following files:

<table>
<thead>
<tr>
<th>Palm Files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobiLink\ListenerSDK\Palm\68k\cw \lib\PalmLsn.lib</td>
<td>Runtime library for Palm Listeners. This provides a message handling routine, Listener controls, and a handler editor.</td>
</tr>
<tr>
<td>MobiLink\ListenerSDK\Palm\68k\cw \rsc\</td>
<td>Contains UI resources for the Palm Listener.</td>
</tr>
<tr>
<td>MobiLink\ListenerSDK\Palm\src \PalmLsn.h</td>
<td>Runtime library header and Palm Listener API.</td>
</tr>
<tr>
<td>MobiLink\ListenerSDK\Palm\src \Treo650.c</td>
<td>Treo 650 implementation.</td>
</tr>
</tbody>
</table>
Message processing interface

The message processing interface is contained in PalmLsn.lib, the Palm Listener Library.

For more information about PalmLsn.lib, see “Palm Listener SDK files” on page 90.

a_palm_msg structure

The Palm Listener SDK uses the a_palm_msg structure to represent Palm Listener messages. The SDK's message processing interface includes functions to allocate and process a_palm_msg instances.

Overview

The following functions can be used for a_palm_msg allocation, message field initialization, and message processing.

- **Message allocation**  You can use the following functions for message allocation and deallocation:
  - “PalmLsnAllocate function” on page 91
  - “PalmLsnFree function” on page 92

- **Message field initialization**  You can use the following functions to assign values to the message, sender, and time fields of an a_palm_msg instance:
  - “PalmLsnDupMessage function” on page 92
  - “PalmLsnDupSender function” on page 93
  - “PalmLsnDupTime function” on page 94

- **Message processing**  You can use the PalmLsnProcess function to process a message's fields and launch an application.
  
  See “PalmLsnProcess function” on page 95.

PalmLsnAllocate function

Returns a new a_palm_msg instance.

**Prototype**

```c
struct a_palm_msg * PalmLsnAllocate( )
```

**Return value**

A new a_palm_msg instance with all fields initialized to zero.

**See also**

- “PalmLsnFree function” on page 92
The following example uses PalmLsnAllocate to allocate an a_palm_msg instance.

```c
a_palm_msg * ulMsg;
// Allocate a message structure
ulMsg = PalmLsnAllocate();
```

**PalmLsnFree function**

Frees message memory resources.

**Prototype**

```c
void PalmLsnFree( struct a_palm_msg * const msg )
```

**Parameters**

- `msg`  
The a_palm_msg instance to be freed.

**See also**

- “Overview” on page 91

**Example**

The following example shows a partial listing for allocating the message structure, processing the message, and using PalmLsnFree to free resources.

```c
a_palm_msg * ulMsg;
...
// Allocate the message structure
ulMsg = PalmLsnAllocate();
...
// Fill the message fields
ret = PalmLsnDupMessage( ulMsg, msgBody );
...
// Process the message
ret = PalmLsnProcess( ulMsg, configDb, NULL, handled );
...
// Free the message
PalmLsnFree( ulMsg );
```

**PalmLsnDupMessage function**

Initializes the message field values of an a_palm_msg instance.

**Prototype**

```c
Err PalmLsnDupMessage(
    struct a_palm_msg * const msg,
```
Char const * message
)

Parameters

● **msg**  
  A pointer to an a_palm_msg instance.

● **message**  
  An input parameter containing the source message text.

Return value

A Palm OS error code. errNone indicates success.

Remarks

The PalmLsnDupMessage function duplicates a text message, extracts the subject, content, and sender fields, and assigns these values to an a_palm_msg instance.

The sender field is not extracted if it does not appear in the message. If you use PalmLsnDupSender it overrides the sender field extracted from PalmLsnDupMessage (if any).

See also

● “PalmLsnDupSender function” on page 93
● “PalmLsnDupTime function” on page 94
● “a_palm_msg structure” on page 91

Example

The following example, used for the Treo 650 smartphone implementation, retrieves a text message and calls PalmLsnDupMessage to initialize the appropriate fields in an a_palm_msg instance.

```c
// Retrieve the entire message body
//
ret = PhnLibGetText( libRef, id, &msgBodyH );
if( ret != errNone ) {
    // handle error
    goto done;
}
msgBody = (Char *)MemHandleLock( msgBodyH );
ret = PalmLsnDupMessage( ulMsg, msgBody );
//
//  msgBodyH must be disposed of by the caller
//
MemHandleUnlock( msgBodyH );
MemHandleFree( msgBodyH );
if( ret != errNone ) {
    // handle error
    goto done;
}
```

**PalmLsnDupSender function**

Initializes the sender field of an a_palm_msg instance.
Prototype

```c
Err PalmLsnDupSender(
    struct a_palm_msg * const msg,
    Char const * sender
)
```

Parameters

- `msg`  A pointer to an a_palm_msg instance.
- `sender`  An input parameter containing the source sender field.

Return value

A Palm OS error code. errNone indicates success.

Remarks

The PalmLsnDupSender function duplicates the sender input parameter and assigns the value to an a_palm_msg instance.

See also

- “PalmLsnDupMessage function” on page 92
- “PalmLsnDupTime function” on page 94
- “a_palm_msg structure” on page 91

**PalmLsnDupTime function**

Initializes the time field of an a_palm_msg instance.

Prototype

```c
Err PalmLsnDupTime(
    struct a_palm_msg * const msg,
    UInt32 const time
)
```

Parameters

- `msg`  A pointer to an a_palm_msg instance.
- `time`  An input parameter containing the source time field.

Return value

A Palm OS error code. errNone indicates success.

Remarks

The PalmLsnDupTime function duplicates the time input parameter and assigns the value to an a_palm_msg instance.

See also

- “PalmLsnDupMessage function” on page 92
- “PalmLsnDupSender function” on page 93
PalmLsnProcess function

Processes a message according to the records in a configuration database.

Prototype

```
palm_lsn_ret PalmLsnProcess(
    struct a_palm_msg * msg,
    Char const * configPDBName,
    UInt16 * const problematicRecNum,
    Boolean * handled
)
```

Parameters

- **msg**  A pointer to an a_palm_msg instance.
- **configPDBName**  A character array containing the name of the configuration database. You can obtain the configuration database name using the PalmLsnGetConfigFileName function.
  
  See “PalmLsnGetConfigFileName” on page 101.
- **problematicRecNum**  An output parameter identifying the index of a problematic or malformed record in the configuration database.
- **handled**  An output parameter indicating if PalmLsnProcess successfully processed the message.

Return value

Return codes defined in the palm_lsn_ret enumeration.

See “palm_lsn_ret enumeration” on page 97.

Remarks

PalmLsnProcess determines the appropriate action to take in response to an incoming message. It compares the message's fields to filters stored in a configuration database.

For more information about creating the Palm Listener configuration database, see “Palm Listener Configuration utility” on page 50.

The records contained in the configuration database store information about message filters and what actions should result from an accepted message.

A configuration record has the following format:

```
[subject=<string>]; [content=<string>];
[message=message_start=<string>]; [sender=<string>];
action=run <app name> [arguments]
```

(arguments) is an application dependent string which may contain action variables.

See also

- “Palm Listener Configuration utility” on page 50
Example

The following is a partial listing used to handle a message. The example allocates the message structure, initializes fields, and processes the message using PalmLsnProcess.

```c
a_palm_msg * ulMsg;
Boolean * handled
Char configDb[ dmDBNameLength ];
...

// Allocate the message structure
ulMsg = PalmLsnAllocate();
...

// Fill the message fields
ret = PalmLsnDupMessage( ulMsg, msgBody );
...

// Get the configuration database name
PalmLsnGetConfigFileName( configDb );

// Process the message
ret = PalmLsnProcess( ulMsg, configDb, NULL, handled );
...

// Free the message
PalmLsnFree( ulMsg );
```

PalmLsnCheckConfigDB function

Reports errors in a Palm Listener configuration database.

Prototype

```c
palm_lsn_ret PalmLsnCheckConfigDB(
    Char const * cfg,
    UInt16 * const rec
)
```

Parameters

- **cfg** A character array containing the name of the configuration database. You can obtain the configuration database name using the PalmLsnGetConfigFileName function.
  
  See “PalmLsnGetConfigFileName” on page 101.
- **rec** An output parameter identifying the index of a problematic or malformed record in the configuration database.

Return value

Return codes defined in the palm_lsn_ret enumeration.
See “palm_lsn_ret enumeration” on page 97.

Remarks
You can use this function to detect errors opening a configuration database or reading its records.

See also
- “PalmLsnProcess function” on page 95

Example
The following example uses PalmLsnCheckConfigDB to detect problematic or malformed records in a configuration database.

```c
Err ret;
UInt16 badRec;
Char configDb[ dmDBNameLength ];

// Get configuration database name
PalmLsnGetConfigFileName( configDb );

// check for errors in the configuration database
ret = PalmLsnCheckConfigDB(configDb, &badRec);
if(ret!=errNone)
{
    // handle error
}
```

palm_lsn_ret enumeration

The palm_lsn_ret enumeration specifies the possible message processing return codes.

Prototype
typedef enum {
    PalmLsnOk = errNone,
    PalmLsnMissingConfig = appErrorClass,
    PalmLsnProblemReadingConfig,
    PalmLsnProblemParsingCmd,
    PalmLsnOutOfMemory,
    PalmLsnUnrecognizedAction,
    PalmLsnRunMissingApp
} palm_lsn_ret;

Parameters

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PalmLsnOk</td>
<td>The function call is successful. This value contains the same value as errNone, a Palm error code indicating no error.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PalmLsnMissingConfig</td>
<td>Indicates a missing Palm Listener configuration database. This field contains the same value as the Palm error code appErrorClass, indicating an application-defined error.</td>
</tr>
<tr>
<td>PalmLsnProblemReadingConfig</td>
<td>Indicates an error reading the Palm Listener configuration database.</td>
</tr>
<tr>
<td>PalmLsnProblemParsingCmd</td>
<td>Indicates an inability to process the command stored in the Palm Listener configuration database.</td>
</tr>
<tr>
<td>PalmLsnOutOfMemory</td>
<td>The function does not run to completion due to an error while allocating memory for message processing.</td>
</tr>
<tr>
<td>PalmLsnUnrecognizedAction</td>
<td>The Listener does not support an action specified in the Palm Listener configuration database.</td>
</tr>
<tr>
<td>PalmLsnRunMissingApp</td>
<td>The Listener cannot launch the application specified in the run action.</td>
</tr>
</tbody>
</table>

See also
- “PalmLsnProcess function” on page 95

**LsnMain function**

Provides the main entry point to *PalmLsn.lib*, the Palm Listener library.

**Prototype**

```c
UInt32 LsnMain(
    UInt16 cmd,
    MemPtr cmdPBP,
    UInt16 launchFlags
)
```

**Parameters**

- **cmd** A Palm OS application launch code.
- **cmdPBP** A pointer to a structure containing launch code parameters. If your application does not have any launch-command-specific parameters, this value is null.
- **launchFlags** Flags that provide extra information about the launch.
Return value

A Palm OS error code. If the Palm listener library successfully processed the launch code, the function returns errNone.

Remarks

The values passed to LsnMain are analogous to the launch code parameters passed to PilotMain, the main entry point of a Palm OS application.

For more information about these parameters, consult your Palm OS Reference.

See also

- “PalmLsnProcess function” on page 95
- “Palm Listener SDK files” on page 90

Example

The following example, used in the Treo 650 smartphone implementation, passes launch code parameters to LsnMain in the main entry point of the Listener application.

```c
UInt32 PilotMain(
    /***************/
    UInt16 cmd,
    MemPtr cmdPBP,
    UInt16 launchFlags )
{
    return( LsnMain( cmd, cmdPBP, launchFlags ) );
}
```
Device dependent functions

You specify device dependent features using a group of functions defined in the Palm Listener SDK. These functions provide:

- **Identification**  You can use the following functions to provide identification information for the Listener and the configuration database:
  
  “PalmLsnTargetCompanyID” on page 100
  “PalmLsnTargetDeviceID” on page 101
  “PalmLsnGetConfigFileName” on page 101

- **Registration or initialization**  You can use the following functions to register or unregister the Listener:
  
  “PalmLsnNormalStart” on page 102
  “PalmLsnNormalStop” on page 102

- **Event handling**  You can use the following function to handle application events:
  
  “PalmLsnNormalHandleEvent” on page 102

  You can use the following function to respond to launch codes which may be device dependent:
  
  “PalmLsnSpecialLaunch” on page 103

---

**PalmLsnTargetCompanyID**

Returns a device's company ID.

**Prototype**

```c
UInt32 PalmLsnTargetCompanyID( )
```

**Return value**

A value containing the ID of the device's company or manufacturer.

**Remarks**

You can use PalmLsnTargetCompanyID and PalmLsnTargetDeviceID to check for device compatibility.

**See also**

- “PalmLsnTargetDeviceID” on page 101

**Example**

The following example, used in the Treo 650 smartphone implementation, returns 'hspr', a company ID for Handspring.

```c
UInt32 PalmLsnTargetCompanyID( void )

/***************************/
```
{    return( 'hspr' );
}

PalmLsnTargetDeviceID

Returns the target device ID.

Prototype
UInt32 PalmLsnTargetDeviceID( )

Return value
A positive integer containing the device ID.

Remarks
You can use PalmLsnTargetCompanyID and PalmLsnTargetDeviceID to check for device compatibility.

See also
● “PalmLsnTargetCompanyID” on page 100

Example
The following example returns the device ID for the Treo 650 simulator.

UInt32 PalmLsnTargetDeviceID( void ) {
    return( kPalmOneDeviceIDTreo650 );
}

PalmLsnGetConfigFileName

Returns a string containing the name of your Palm Listener configuration database.

Prototype
void PalmLsnGetConfigFileName( Char * configPDBName )

Parameters
● configPDBName An output parameter containing the name of your Palm Listener configuration database.

Remarks
You can use this function to obtain the configuration database file name to pass into PalmLsnProcess.

To use the default configuration database file name lsncfg copy PalmLsnDefaultConfigDB (defined in PalmLsn.h) into the output parameter.

See also
● “PalmLsnProcess function” on page 95
Example
The following example, used for the Treo 650 smartphone implementation, returns the default configuration database name in the output parameter.

```c
void PalmLsnGetConfigFileName( Char * configPDBName )
{
    StrCopy( configPDBName, PalmLsnDefaultConfigDB );
}
```

PalmLsnNormalStart
Provides custom actions when your Listener application starts.

Prototype
```
Err PalmLsnNormalStart( )
```

Return value
A Palm OS error code. errNone indicates success.

Remarks
PalmLsnNormalStart provides a means to register your Listener device.

See also
● “PalmLsnNormalStop” on page 102  
● “PalmLsnSpecialLaunch” on page 103

PalmLsnNormalStop
Provides custom actions when your Listener application exits from the event loop.

Prototype
```
void PalmLsnNormalStop( )
```

Remarks
If you want to continue listening, do not unregister your device in PalmLsnNormalStop. You can also use this function to get and set the current application preferences.

See also
● “PalmLsnNormalStart” on page 102

PalmLsnNormalHandleEvent
Handles application events.
Prototype

Boolean PalmLsnNormalHandleEvent( EventPtr eventP )

Parameters

- **eventP** A pointer to an application event.

Return value

Returns true if the event was handled.

Remarks

You can use this function to handle application events.

**PalmLsnSpecialLaunch**

Responds to launch codes which may be device dependent.

Prototype

Err PalmLsnSpecialLaunch(
    UInt16  cmd,
    MemPtr  cmdPBP,
    UInt16  launchFlags
)

Parameters

- **cmd** The Palm OS application launch code.
- **cmdPBP** A pointer to a structure containing launch code parameters. If your application does not have any launch-command-specific parameters, this value is null.
- **launchFlags** Flags that indicate status information about your application.

Return value

A Palm OS error code. errNone indicates success.

Remarks

This function responds to device dependent or standard launch codes not defined as sysAppLaunchCmdNormalLaunch.

Example

The following example, used for the Treo 650 smartphone implementation, uses PalmLsnSpecialLaunch to handle Listener events.

```c
Err PalmLsnSpecialLaunch( UInt16 cmd, MemPtr cmdPBP, UInt16 /*launchFlags*/ )
{
    switch( cmd ) {
    case sysAppLaunchCmdSystemReset:
        // Fall through
    case sysAppLaunchCmdSyncNotify:
        // Fall through
    case phnLibLaunchCmdRegister:
```
If a message is detected, handleMessage is used to process the message into the appropriate action.

```c
static Err handleMessage( PhnDatabaseID id, Boolean * handled )
{ /* This routine will construct a_palm_msg and then call /PalmLsnProcess to process it. */

    a_palm_msg *     ulMsg;
    Err              ret;
    Boolean          newlyLoaded;
    PhnAddressList   addrList;
    PhnAddressHandle addrH;
    MemHandle        msgBodyH;
    Char *           msgSender;
    Char *           msgBody;
    UInt32           msgTime;
    char            configDb[ dmDBNameLength ];
    uint16           libRef = 0;
    // CDMA workaround recommended by Handspring
    DmOpenRef        openRef = 0;

    *handled = false;

    // Allocate a message structure for passing over
    // to PalmLsnProcess later

    ulMsg = PalmLsnAllocate();
    if( ulMsg == NULL ) { return( sysErrNoFreeRAM ); }

    // Load the phone library

    ret = findOrLoadPhoneLibrary( &libRef, &newlyLoaded );
    if( ret != errNone ) { goto done; }

    openRef = PhnLibGetDBRef( libRef );
    // Retrieve sender of the message

    ret = PhnLibGetAddresses( libRef, id, &addrList );
    if( ret != errNone ) { goto done; }

    ret = PhnLibGetNth( libRef, addrList, 1, &addrH );
    if( ret != errNone ) { PhnLibDisposeAddressList( libRef, addrList );
```
goto done;
}

msgSender = PhnLibGetField( libRef, addrH, phnAddrFldPhone );
if( msgSender != NULL ) {
    ret = PalmLsnDupSender( ulMsg, msgSender );
    MemPtrFree( msgSender );
}
PhnLibDisposeAddressList( libRef, addrList );
if( ret != errNone ) {
    goto done;
}

// Retrieve message time
ret = PhnLibGetDate( libRef, id, &msgTime );
if( ret != errNone ) {
    goto done;
}
ret = PalmLsnDupTime( ulMsg, msgTime );
if( ret != errNone ) {
    goto done;
}

// Retrieve the entire message body
ret = PhnLibGetText( libRef, id, &msgBodyH );
if( ret != errNone ) {
    goto done;
}
msgBody = (Char *)MemHandleLock( msgBodyH );
ret = PalmLsnDupMessage( ulMsg, msgBody );

// msgBodyH must be disposed of by the caller
MemHandleUnlock( msgBodyH );
MemHandleFree( msgBodyH );
if( ret != errNone ) {
    goto done;
}

// Get the configuration database name
PalmLsnGetConfigFileName( configDb );

// Call PalmLsnProcess to process the message
ret = PalmLsnProcess( ulMsg, configDb, NULL, handled );
done:
    if( ulMsg != NULL ) {
        PalmLsnFree( ulMsg );
    }
    PhnLibReleaseDBRef( libRef, openRef );

// Unload the phone library before any possible application switch
if( newlyLoaded ) {
    unloadPhoneLibrary( libRef );
    newlyLoaded = false;
}
return( ret );
}
Part I. Glossary
Adaptive Server Anywhere (ASA)

The relational database server component of SQL Anywhere Studio, intended for use in mobile and embedded environments or as a server for small and medium-sized businesses. In version 10.0.0, Adaptive Server Anywhere was renamed SQL Anywhere Server, and SQL Anywhere Studio was renamed SQL Anywhere.

See also: “SQL Anywhere” on page 132.

agent ID

See also: “client message store ID” on page 111.

article

In MobiLink or SQL Remote, an article is a database object that represents a whole table, or a subset of the columns and rows in a table. Articles are grouped together in a publication.

See also:
● “replication” on page 130
● “publication” on page 127

atomic transaction

A transaction that is guaranteed to complete successfully or not at all. If an error prevents part of an atomic transaction from completing, the transaction is rolled back to prevent the database from being left in an inconsistent state.

base table

Permanent tables for data. Tables are sometimes called base tables to distinguish them from temporary tables and views.

See also:
● “temporary table” on page 134
● “view” on page 136

bit array

A bit array is a type of array data structure that is used for efficient storage of a sequence of bits. A bit array is similar to a character string, except that the individual pieces are 0s (zeros) and 1s (ones) instead of characters. Bit arrays are typically used to hold a string of Boolean values.
business rule

A guideline based on real-world requirements. Business rules are typically implemented through check constraints, user-defined data types, and the appropriate use of transactions.

See also:
- “constraint” on page 112
- “user-defined data type” on page 136

carrier

A MobiLink object, stored in MobiLink system tables or a Notifier properties file, that contains information about a public carrier for use by server-initiated synchronization.

See also: “server-initiated synchronization” on page 132.

character set

A character set is a set of symbols, including letters, digits, spaces, and other symbols. An example of a character set is ISO-8859-1, also known as Latin1.

See also:
- “code page” on page 111
- “encoding” on page 116
- “collation” on page 111

check constraint

A restriction that enforces specified conditions on a column or set of columns.

See also:
- “constraint” on page 112
- “foreign key constraint” on page 118
- “primary key constraint” on page 127
- “unique constraint” on page 135

checkpoint

The point at which all changes to the database are saved to the database file. At other times, committed changes are saved only to the transaction log.

checksum

The calculated number of bits of a database page that is recorded with the database page itself. The checksum allows the database management system to validate the integrity of the page by ensuring that the numbers match as the page is being written to disk. If the counts match, it's assumed that page was successfully written.

client message store

In QAnywhere, a SQL Anywhere database on the remote device that stores messages.
client message store ID

In QAnywhere, a MobiLink remote ID that uniquely identifies a client message store.

client/server

A software architecture where one application (the client) obtains information from and sends information to another application (the server). The two applications often reside on different computers connected by a network.

code page

A code page is an encoding that maps characters of a character set to numeric representations, typically an integer between 0 and 255. An example of a code page is Windows code page 1252. For the purposes of this documentation, code page and encoding are interchangeable terms.

See also:

- “character set” on page 110
- “encoding” on page 116
- “collation” on page 111

collation

A combination of a character set and a sort order that defines the properties of text in the database. For SQL Anywhere databases, the default collation is determined by the operating system and language on which the server is running; for example, the default collation on English Windows systems is 1252LATIN1. A collation, also called a collating sequence, is used for comparing and sorting strings.

See also:

- “character set” on page 110
- “code page” on page 111
- “encoding” on page 116

command file

A text file containing SQL statements. Command files can be built manually, or they can be built automatically by database utilities. The dbunload utility, for example, creates a command file consisting of the SQL statements necessary to recreate a given database.

communication stream

In MobiLink, the network protocol used for communication between the MobiLink client and the MobiLink server.

concurrency

The simultaneous execution of two or more independent, and possibly competing, processes. SQL Anywhere automatically uses locking to isolate transactions and ensure that each concurrent application sees a consistent set of data.
See also:

- “transaction” on page 134
- “isolation level” on page 120

**conflict resolution**

In MobiLink, conflict resolution is logic that specifies what to do when two users modify the same row on different remote databases.

**connection ID**

A unique number that identifies a given connection between a client application and the database. You can determine the current connection ID using the following SQL statement:

```
SELECT CONNECTION_PROPERTY( 'Number' );
```

**connection-initiated synchronization**

A form of MobiLink server-initiated synchronization in which synchronization is initiated when there are changes to connectivity.

See also: “server-initiated synchronization” on page 132.

**connection profile**

A set of parameters that are required to connect to a database, such as user name, password, and server name, that is stored and used as a convenience.

**consolidated database**

In distributed database environments, a database that stores the master copy of the data. In case of conflict or discrepancy, the consolidated database is considered to have the primary copy of the data.

See also:

- “synchronization” on page 134
- “replication” on page 130

**constraint**

A restriction on the values contained in a particular database object, such as a table or column. For example, a column may have a uniqueness constraint, which requires that all values in the column be different. A table may have a foreign key constraint, which specifies how the information in the table relates to data in some other table.

See also:

- “check constraint” on page 110
- “foreign key constraint” on page 118
- “primary key constraint” on page 127
- “unique constraint” on page 135
contention

The act of competing for resources. For example, in database terms, two or more users trying to edit the
same row of a database contend for the rights to edit that row.

correlation name

The name of a table or view that is used in the FROM clause of a query—either its original name, or an
alternate name, that is defined in the FROM clause.

creator ID

In UltraLite Palm OS applications, an ID that is assigned when the application is created.

cursor

A named linkage to a result set, used to access and update rows from a programming interface. In SQL
Anywhere, cursors support forward and backward movement through the query results. Cursors consist of
two parts: the cursor result set, typically defined by a SELECT statement; and the cursor position.

See also:
  ● “cursor result set” on page 113
  ● “cursor position” on page 113

cursor position

A pointer to one row within the cursor result set.

See also:
  ● “cursor” on page 113
  ● “cursor result set” on page 113

cursor result set

The set of rows resulting from a query that is associated with a cursor.

See also:
  ● “cursor” on page 113
  ● “cursor position” on page 113

data cube

A multi-dimensional result set with each dimension reflecting a different way to group and sort the same
results. Data cubes provide complex information about data that would otherwise require self-join queries
and correlated subqueries. Data cubes are a part of OLAP functionality.

data definition language (DDL)

The subset of SQL statements for defining the structure of data in the database. DDL statements create,
modify, and remove database objects, such as tables and users.
data manipulation language (DML)

The subset of SQL statements for manipulating data in the database. DML statements retrieve, insert, update, and delete data in the database.

data type

The format of data, such as CHAR or NUMERIC. In the ANSI SQL standard, data types can also include a restriction on size, character set, and collation.

See also: “domain” on page 116.

database

A collection of tables that are related by primary and foreign keys. The tables hold the information in the database. The tables and keys together define the structure of the database. A database management system accesses this information.

See also:
- “foreign key” on page 117
- “primary key” on page 127
- “database management system (DBMS)” on page 114
- “relational database management system (RDBMS)” on page 129

database administrator (DBA)

The user with the permissions required to maintain the database. The DBA is generally responsible for all changes to a database schema, and for managing users and groups. The role of database administrator is automatically built into databases as user ID DBA with password sql.

database connection

A communication channel between a client application and the database. A valid user ID and password are required to establish a connection. The privileges granted to the user ID determine the actions that can be carried out during the connection.

database file

A database is held in one or more database files. There is an initial file, and subsequent files are called dbspaces. Each table, including its indexes, must be contained within a single database file.

See also: “dbspace” on page 115.

database management system (DBMS)

A collection of programs that allow you to create and use databases.

See also: “relational database management system (RDBMS)” on page 129.
**database name**

The name given to a database when it is loaded by a server. The default database name is the root of the initial database file.

See also: “database file” on page 114.

**database object**

A component of a database that contains or receives information. Tables, indexes, views, procedures, and triggers are database objects.

**database owner (dbo)**

A special user that owns the system objects not owned by SYS.

See also:
- “database administrator (DBA)” on page 114
- “SYS” on page 134

**database server**

A computer program that regulates all access to information in a database. SQL Anywhere provides two types of servers: network servers and personal servers.

**DBA authority**

The level of permission that enables a user to do administrative activity in the database. The DBA user has DBA authority by default.

See also: “database administrator (DBA)” on page 114.

**dbspace**

An additional database file that creates more space for data. A database can be held in up to 13 separate files (an initial file and 12 dbspaces). Each table, together with its indexes, must be contained in a single database file. The SQL command CREATE DBSPACE adds a new file to the database.

See also: “database file” on page 114.

**deadlock**

A state where a set of transactions arrives at a place where none can proceed.

**device tracking**

Functionality in MobiLink server-initiated synchronization that allows you to address messages using the MobiLink user name that identifies a remote device.

See also: “server-initiated synchronization” on page 132.
direct row handling

In MobiLink, a way to synchronize table data to sources other than the MobiLink-supported consolidated databases. You can implement both uploads and downloads with direct row handling.

See also:

- “consolidated database” on page 112
- “SQL-based synchronization” on page 132

domain

Aliases for built-in data types, including precision and scale values where applicable, and optionally including DEFAULT values and CHECK conditions. Some domains, such as the monetary data types, are pre-defined in SQL Anywhere. Also called user-defined data type.

See also: “data type” on page 114.
download

The stage in synchronization where data is transferred from the consolidated database to a remote database.
dynamic SQL

SQL that is generated programmatically by your program before it is executed. UltraLite dynamic SQL is a variant designed for small-footprint devices.

EBF

Express Bug Fix. An express bug fix is a subset of the software with one or more bug fixes. The bug fixes are listed in the release notes for the update. Bug fix updates may only be applied to installed software with the same version number. Some testing has been performed on the software, but the software has not undergone full testing. You should not distribute these files with your application unless you have verified the suitability of the software yourself.

embedded SQL

A programming interface for C programs. SQL Anywhere embedded SQL is an implementation of the ANSI and IBM standard.

encoding

Also known as character encoding, an encoding is a method by which each character in a character set is mapped onto one or more bytes of information, typically represented as a hexadecimal number. An example of an encoding is UTF-8.

See also:

- “character set” on page 110
- “code page” on page 111
- “collation” on page 111
**event model**

In MobiLink, the sequence of events that make up a synchronization, such as begin_synchronization and download_cursor. Events are invoked if a script is created for them.

**external login**

An alternate login name and password used when communicating with a remote server. By default, SQL Anywhere uses the names and passwords of its clients whenever it connects to a remote server on behalf of those clients. However, this default can be overridden by creating external logins. External logins are alternate login names and passwords used when communicating with a remote server.

**extraction**

In SQL Remote replication, the act of unloading the appropriate structure and data from the consolidated database. This information is used to initialize the remote database.

See also: “replication” on page 130.

**failover**

Switching to a redundant or standby server, system, or network on failure or unplanned termination of the active server, system, or network. Failover happens automatically.

**FILE**

In SQL Remote replication, a message system that uses shared files for exchanging replication messages. This is useful for testing and for installations without an explicit message-transport system.

See also: “replication” on page 130.

**file-based download**

In MobiLink, a way to synchronize data in which downloads are distributed as files, allowing offline distribution of synchronization changes.

**file-definition database**

In MobiLink, a SQL Anywhere database that is used for creating download files.

See also: “file-based download” on page 117.

**foreign key**

One or more columns in a table that duplicate the primary key values in another table. Foreign keys establish relationships between tables.

See also:
- “primary key” on page 127
- “foreign table” on page 118
foreign key constraint

A restriction on a column or set of columns that specifies how the data in the table relates to the data in some other table. Imposing a foreign key constraint on a set of columns makes those columns the foreign key.

See also:
- “constraint” on page 112
- “check constraint” on page 110
- “primary key constraint” on page 127
- “unique constraint” on page 135

foreign table

The table containing the foreign key.

See also: “foreign key” on page 117.

full backup

A backup of the entire database, and optionally, the transaction log. A full backup contains all the information in the database and thus provides protection in the event of a system or media failure.

See also: “incremental backup” on page 119.

gateway

A MobiLink object, stored in MobiLink system tables or a Notifier properties file, that contains information about how to send messages for server-initiated synchronization.

See also: “server-initiated synchronization” on page 132.

generated join condition

A restriction on join results that is automatically generated. There are two types: key and natural. Key joins are generated when you specify KEY JOIN or when you specify the keyword JOIN but do not use the keywords CROSS, NATURAL, or ON. For a key join, the generated join condition is based on foreign key relationships between tables. Natural joins are generated when you specify NATURAL JOIN; the generated join condition is based on common column names in the two tables.

See also:
- “join” on page 121
- “join condition” on page 121

generation number

In MobiLink, a mechanism for forcing remote databases to upload data before applying any more download files.

See also: “file-based download” on page 117.
global temporary table

A type of temporary table for which data definitions are visible to all users until explicitly dropped. Global temporary tables let each user open their own identical instance of a table. By default, rows are deleted on commit, and rows are always deleted when the connection is ended.

See also:

● “temporary table” on page 134
● “local temporary table” on page 122

grant option

The level of permission that allows a user to grant permissions to other users.

hash

A hash is an index optimization that transforms index entries into keys. An index hash aims to avoid the expensive operation of finding, loading, and then unpacking the rows to determine the indexed value, by including enough of the actual row data with its row ID.

histogram

The most important component of column statistics, histograms are a representation of data distribution. SQL Anywhere maintains histograms to provide the optimizer with statistical information about the distribution of values in columns.

iAnywhere JDBC driver

The iAnywhere JDBC driver provides a JDBC driver that has some performance benefits and feature benefits compared to the pure Java jConnect JDBC driver, but which is not a pure-Java solution. The iAnywhere JDBC driver is recommended in most cases.

See also:

● “JDBC” on page 121
● “jConnect” on page 121

identifier

A string of characters used to reference a database object, such as a table or column. An identifier may contain any character from A through Z, a through z, 0 through 9, underscore (_), at sign (@), number sign (#), or dollar sign ($).

incremental backup

A backup of the transaction log only, typically used between full backups.

See also: “transaction log” on page 135.
index

A sorted set of keys and pointers associated with one or more columns in a base table. An index on one or more columns of a table can improve performance.

InfoMaker

A reporting and data maintenance tool that lets you create sophisticated forms, reports, graphs, cross-tabs, and tables, as well as applications that use these reports as building blocks.

inner join

A join in which rows appear in the result set only if both tables satisfy the join condition. Inner joins are the default.

See also:
- “join” on page 121
- “outer join” on page 125

integrated login

A login feature that allows the same single user ID and password to be used for operating system logins, network logins, and database connections.

integrity

Adherence to rules that ensure that data is correct and accurate, and that the relational structure of the database is intact.

See also: “referential integrity” on page 129.

Interactive SQL

A SQL Anywhere application that allows you to query and alter data in your database, and modify the structure of your database. Interactive SQL provides a pane for you to enter SQL statements, as well as panes that return information about how the query was processed and the result set.

isolation level

The degree to which operations in one transaction are visible to operations in other concurrent transactions. There are four isolation levels, numbered 0 through 3. Level 3 provides the highest level of isolation. Level 0 is the default setting. SQL Anywhere also supports three snapshot isolation levels: snapshot, statement-snapshot, and readonly-statement-snapshot.

See also: “snapshot isolation” on page 132.

JAR file

Java archive file. A compressed file format consisting of a collection of one or more packages used for Java applications. It includes all the resources necessary to install and run a Java program in a single compressed file.
Java class

The main structural unit of code in Java. It is a collection of procedures and variables grouped together because they all relate to a specific, identifiable category.

jConnect

A Java implementation of the JavaSoft JDBC standard. It provides Java developers with native database access in multi-tier and heterogeneous environments. However, the iAnywhere JDBC driver is the preferred JDBC driver for most cases.

See also:
- “JDBC” on page 121
- “iAnywhere JDBC driver” on page 119

JDBC

Java Database Connectivity. A SQL-language programming interface that allows Java applications to access relational data. The preferred JDBC driver is the iAnywhere JDBC driver.

See also:
- “jConnect” on page 121
- “iAnywhere JDBC driver” on page 119

join

A basic operation in a relational system that links the rows in two or more tables by comparing the values in specified columns.

join condition

A restriction that affects join results. You specify a join condition by inserting an ON clause or WHERE clause immediately after the join. In the case of natural and key joins, SQL Anywhere generates a join condition.

See also:
- “join” on page 121
- “generated join condition” on page 118

join type

SQL Anywhere provides four types of joins: cross join, key join, natural join, and joins using an ON clause.

See also: “join” on page 121.

Listener

A program, dblsn, that is used by MobiLink server-initiated synchronization. Listeners are installed on remote devices and configured to initiate actions on the device when they receive information from a Notifier.
local temporary table

A type of temporary table that exists only for the duration of a compound statement or until the end of the connection. Local temporary tables are useful when you need to load a set of data only once. By default, rows are deleted on commit.

See also:
- “temporary table” on page 134
- “global temporary table” on page 119

lock

A concurrency control mechanism that protects the integrity of data during the simultaneous execution of multiple transactions. SQL Anywhere automatically applies locks to prevent two connections from changing the same data at the same time, and to prevent other connections from reading data that is in the process of being changed.

You control locking by setting the isolation level.

See also:
- “isolation level” on page 120
- “concurrency” on page 111
- “integrity” on page 120

log file

A log of transactions maintained by SQL Anywhere. The log file is used to ensure that the database is recoverable in the event of a system or media failure, to improve database performance, and to allow data replication using SQL Remote.

See also:
- “transaction log” on page 135
- “transaction log mirror” on page 135
- “full backup” on page 118

logical index

A reference (pointer) to a physical index. There is no indexing structure stored on disk for a logical index.

LTM

Log Transfer Manager (LTM) also called Replication Agent. Used with Replication Server, the LTM is the program that reads a database transaction log and sends committed changes to Sybase Replication Server.

See: “Replication Server” on page 130.
**maintenance release**

A maintenance release is a complete set of software that upgrades installed software from an older version with the same major version number (version number format is `major.minor.patch.build`). Bug fixes and other changes are listed in the release notes for the upgrade.

**materialized view**

A materialized view is a view that has been computed and stored on disk. Materialized views have characteristics of both views (they are defined using a query specification), and of tables (they allow most table operations to be performed on them).

See also:
- “base table” on page 109
- “view” on page 136

**message log**

A log where messages from an application such as a database server or MobiLink server can be stored. This information can also appear in a messages window or be logged to a file. The message log includes informational messages, errors, warnings, and messages from the MESSAGE statement.

**message store**

In QAnywhere, databases on the client and server device that store messages.

See also:
- “client message store” on page 110
- “server message store” on page 132

**message system**

In SQL Remote replication, a protocol for exchanging messages between the consolidated database and a remote database. SQL Anywhere includes support for the following message systems: FILE, FTP, and SMTP.

See also:
- “replication” on page 130
- “FILE” on page 117

**message type**

In SQL Remote replication, a database object that specifies how remote users communicate with the publisher of a consolidated database. A consolidated database may have several message types defined for it; this allows different remote users to communicate with it using different message systems.

See also:
- “replication” on page 130
- “consolidated database” on page 112
metadata

Data about data. Metadata describes the nature and content of other data.

See also: “schema” on page 131.

mirror log

See also: “transaction log mirror” on page 135.

MobiLink

A session-based synchronization technology designed to synchronize UltraLite and SQL Anywhere remote databases with a consolidated database.

See also:

- “consolidated database” on page 112
- “synchronization” on page 134
- “UltraLite” on page 135

MobiLink client

There are two kinds of MobiLink clients. For SQL Anywhere remote databases, the MobiLink client is the dbmlsync command line utility. For UltraLite remote databases, the MobiLink client is built in to the UltraLite runtime library.

MobiLink Monitor

A graphical tool for monitoring MobiLink synchronizations.

MobiLink server

The computer program that runs MobiLink synchronization, mlsrv11.

MobiLink system table

System tables that are required by MobiLink synchronization. They are installed by MobiLink setup scripts into the MobiLink consolidated database.

MobiLink user

A MobiLink user is used to connect to the MobiLink server. You create the MobiLink user on the remote database and register it in the consolidated database. MobiLink user names are entirely independent of database user names.

network protocol

The type of communication, such as TCP/IP or HTTP.
network server

A database server that accepts connections from computers sharing a common network.

See also: “personal server” on page 126.

normalization

The refinement of a database structure to eliminate redundancy and improve organization according to rules based on relational database theory.

Notifier

A program that is used by MobiLink server-initiated synchronization. Notifiers run on the same computer as the MobiLink server. They poll the consolidated database for push requests, and then send notifications to Listeners.

See also:

- “server-initiated synchronization” on page 132
- “Listener” on page 121

object tree

In Sybase Central, the hierarchy of database objects. The top level of the object tree shows all products that your version of Sybase Central supports. Each product expands to reveal its own sub-tree of objects.

See also: “Sybase Central” on page 134.

ODBC

Open Database Connectivity. A standard Windows interface to database management systems. ODBC is one of several interfaces supported by SQL Anywhere.

ODBC Administrator

A Microsoft program included with Windows operating systems for setting up ODBC data sources.

ODBC data source

A specification of the data a user wants to access via ODBC, and the information needed to get to that data.

outer join

A join that preserves all the rows in a table. SQL Anywhere supports left, right, and full outer joins. A left outer join preserves the rows in the table to the left of the join operator, and returns a null when a row in the right table does not satisfy the join condition. A full outer join preserves all the rows from both tables.

See also:

- “join” on page 121
- “inner join” on page 120
package

In Java, a collection of related classes.

parse tree

An algebraic representation of a query.

PDB

A Palm database file.

performance statistic

A value reflecting the performance of the database system. The CURRREAD statistic, for example, represents the number of file reads issued by the engine that have not yet completed.

personal server

A database server that runs on the same computer as the client application. A personal database server is typically used by a single user on a single computer, but it can support several concurrent connections from that user.

physical index

The actual indexing structure of an index, as it is stored on disk.

plug-in module

In Sybase Central, a way to access and administer a product. Plug-ins are usually installed and registered automatically with Sybase Central when you install the respective product. Typically, a plug-in appears as a top-level container, in the Sybase Central main window, using the name of the product itself; for example, SQL Anywhere.

See also: “Sybase Central” on page 134.

policy

In QAnywhere, the way you specify when message transmission should occur.

polling

In MobiLink server-initiated synchronization, the way the Notifier detects push requests on the consolidated database.

See also: “server-initiated synchronization” on page 132.

PowerDesigner

A database modeling application. PowerDesigner provides a structured approach to designing a database or data warehouse. SQL Anywhere includes the Physical Data Model component of PowerDesigner.
PowerJ

A Sybase product for developing Java applications.

predicate

A conditional expression that is optionally combined with the logical operators AND and OR to make up the set of conditions in a WHERE or HAVING clause. In SQL, a predicate that evaluates to UNKNOWN is interpreted as FALSE.

primary key

A column or list of columns whose values uniquely identify every row in the table.

See also: “foreign key” on page 117.

primary key constraint

A uniqueness constraint on the primary key columns. A table can have only one primary key constraint.

See also:
- “constraint” on page 112
- “check constraint” on page 110
- “foreign key constraint” on page 118
- “unique constraint” on page 135
- “integrity” on page 120

primary table

The table containing the primary key in a foreign key relationship.

proxy table

A local table containing metadata used to access a table on a remote database server as if it were a local table.

See also: “metadata” on page 124.

publication

In MobiLink or SQL Remote, a database object that identifies data that is to be synchronized. In MobiLink, publications exist only on the clients. A publication consists of articles. SQL Remote users can receive a publication by subscribing to it. MobiLink users can synchronize a publication by creating a synchronization subscription to it.

See also:
- “replication” on page 130
- “article” on page 109
- “publication update” on page 128
**publication update**

In SQL Remote replication, a list of changes made to one or more publications in one database. A publication update is sent periodically as part of a replication message to the remote database(s).

See also:
- “replication” on page 130
- “publication” on page 127

**publisher**

In SQL Remote replication, the single user in a database who can exchange replication messages with other replicating databases.

See also: “replication” on page 130.

**push notification**

In QAnywhere, a special message delivered from the server to a QAnywhere client that prompts the client to initiate a message transmission.

See also: “QAnywhere” on page 128.

**push request**

In MobiLink server-initiated synchronization, a row in a SQL result set or table on the consolidated database that contains a notification and information about how to send the notification.

See also: “server-initiated synchronization” on page 132.

**QAnywhere**

Application-to-application messaging, including mobile device to mobile device and mobile device to and from the enterprise, that permits communication between custom programs running on mobile or wireless devices and a centrally located server application.

**QAnywhere agent**

In QAnywhere, a process running on the client device that monitors the client message store and determines when message transmission should occur.

**query**

A SQL statement or group of SQL statements that access and/or manipulate data in a database.

See also: “SQL” on page 132.

**Redirector**

A web server plug-in that routes requests and responses between a client and the MobiLink server. This plug-in also implements load-balancing and failover mechanisms.
reference database

In MobiLink, a SQL Anywhere database used in the development of UltraLite clients. You can use a single SQL Anywhere database as both reference and consolidated database during development. Databases made with other products cannot be used as reference databases.

referencing object

An object, such as a view, whose definition directly references another object in the database, such as a table.

See also: “foreign key” on page 117.

referenced object

An object, such as a table, that is directly referenced in the definition of another object, such as a view.

See also: “primary key” on page 127.

referential integrity

Adherence to rules governing data consistency, specifically the relationships between the primary and foreign key values in different tables. To have referential integrity, the values in each foreign key must correspond to the primary key values of a row in the referenced table.

See also:
- “primary key” on page 127
- “foreign key” on page 117

regular expression

A regular expression is a sequence of characters, wildcards, and operators that defines a pattern to search for within a string.

relational database management system (RDBMS)

A type of database management system that stores data in the form of related tables.

See also: “database management system (DBMS)” on page 114.

remote database

In MobiLink or SQL Remote, a database that exchanges data with a consolidated database. Remote databases may share all or some of the data in the consolidated database.

See also:
- “synchronization” on page 134
- “consolidated database” on page 112
remote DBA authority

In SQL Remote, a level of permission required by the Message Agent. In MobiLink, a level of permission required by the SQL Anywhere synchronization client (dbmlsync). When the Message Agent or synchronization client connects as a user who has this authority, it has full DBA access. The user ID has no additional permissions when not connected through the Message Agent or synchronization client.

See also: “DBA authority” on page 115.

remote ID

A unique identifier in SQL Anywhere and UltraLite databases that is used by MobiLink. The remote ID is initially set to NULL and is set to a GUID during a database's first synchronization.

replication

The sharing of data among physically distinct databases. Sybase has three replication technologies: MobiLink, SQL Remote, and Replication Server.

Replication Agent

See: “LTM” on page 122.

replication frequency

In SQL Remote replication, a setting for each remote user that determines how often the publisher's message agent should send replication messages to that remote user.

See also: “replication” on page 130.

replication message

In SQL Remote or Replication Server, a communication sent between a publishing database and a subscribing database. Messages contain data, passthrough statements, and information required by the replication system.

See also:
- “replication” on page 130
- “publication update” on page 128

Replication Server

A Sybase connection-based replication technology that works with SQL Anywhere and Adaptive Server Enterprise. It is intended for near-real time replication between a small number of databases.

See also: “LTM” on page 122.

role

In conceptual database modeling, a verb or phrase that describes a relationship from one point of view. You can describe each relationship with two roles. Examples of roles are "contains" and "is a member of."
role name

The name of a foreign key. This is called a role name because it names the relationship between the foreign table and primary table. By default, the role name is the table name, unless another foreign key is already using that name, in which case the default role name is the table name followed by a three-digit unique number. You can also create the role name yourself.

See also: “foreign key” on page 117.

rollback log

A record of the changes made during each uncommitted transaction. In the event of a ROLLBACK request or a system failure, uncommitted transactions are reversed out of the database, returning the database to its former state. Each transaction has a separate rollback log, which is deleted when the transaction is complete.

See also: “transaction” on page 134.

row-level trigger

A trigger that executes once for each row that is changed.

See also:

- “trigger” on page 135
- “statement-level trigger” on page 133

schema

The structure of a database, including tables, columns, and indexes, and the relationships between them.

script

In MobiLink, code written to handle MobiLink events. Scripts programmatically control data exchange to meet business needs.

See also: “event model” on page 117.

script-based upload

In MobiLink, a way to customize the upload process as an alternative to using the log file.

script version

In MobiLink, a set of synchronization scripts that are applied together to create a synchronization.

secured feature

A feature specified by the -sf option when a database server is started, so it is not available for any database running on that database server.
server-initiated synchronization

A way to initiate MobiLink synchronization programmatically from the consolidated database.

server management request

A QAnywhere message that is formatted as XML and sent to the QAnywhere system queue as a way to administer the sever message store or monitor QAnywhere applications.

server message store

In QAnywhere, a relational database on the server that temporarily stores messages until they are transmitted to a client message store or JMS system. Messages are exchanged between clients via the server message store.

service

In Windows operating systems, a way of running applications when the user ID running the application is not logged on.

session-based synchronization

A type of synchronization where synchronization results in consistent data representation across both the consolidated and remote databases. MobiLink is session-based.

snapshot isolation

A type of isolation level that returns a committed version of the data for transactions that issue read requests. SQL Anywhere provides three snapshot isolation levels: snapshot, statement-snapshot, and readonly-statement-snapshot. When using snapshot isolation, read operations do not block write operations.

See also: “isolation level” on page 120.

SQL

The language used to communicate with relational databases. ANSI has defined standards for SQL, the latest of which is SQL-2003. SQL stands, unofficially, for Structured Query Language.

SQL Anywhere

The relational database server component of SQL Anywhere that is intended for use in mobile and embedded environments or as a server for small and medium-sized businesses. SQL Anywhere is also the name of the package that contains the SQL Anywhere RDBMS, the UltraLite RDBMS, MobiLink synchronization software, and other components.

SQL-based synchronization

In MobiLink, a way to synchronize table data to MobiLink-supported consolidated databases using MobiLink events. For SQL-based synchronization, you can use SQL directly or you can return SQL using the MobiLink server APIs for Java and .NET.
SQL Remote

A message-based data replication technology for two-way replication between consolidated and remote databases. The consolidated and remote databases must be SQL Anywhere.

SQL statement

A string containing SQL keywords designed for passing instructions to a DBMS.

See also:
- “schema” on page 131
- “SQL” on page 132
- “database management system (DBMS)” on page 114

statement-level trigger

A trigger that executes after the entire triggering statement is completed.

See also:
- “trigger” on page 135
- “row-level trigger” on page 131

stored procedure

A program comprised of a sequence of SQL instructions, stored in the database and used to perform a particular task.

string literal

A string literal is a sequence of characters enclosed in single quotes.

subquery

A SELECT statement that is nested inside another SELECT, INSERT, UPDATE, or DELETE statement, or another subquery.

There are two types of subquery: correlated and nested.

subscription

In MobiLink synchronization, a link in a client database between a publication and a MobiLink user, allowing the data described by the publication to be synchronized.

In SQL Remote replication, a link between a publication and a remote user, allowing the user to exchange updates on that publication with the consolidated database.

See also:
- “publication” on page 127
- “MobiLink user” on page 124

Copyright © 2008, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.0 133
Sybase Central

A database management tool that provides SQL Anywhere database settings, properties, and utilities in a graphical user interface. Sybase Central can also be used for managing other Sybase products, including MobiLink.

synchronization

The process of replicating data between databases using MobiLink technology.

In SQL Remote, synchronization is used exclusively to denote the process of initializing a remote database with an initial set of data.

See also:

- “MobiLink” on page 124
- “SQL Remote” on page 133

SYS

A special user that owns most of the system objects. You cannot log in as SYS.

system object

Database objects owned by SYS or dbo.

system table

A table, owned by SYS or dbo, that holds metadata. System tables, also known as data dictionary tables, are created and maintained by the database server.

system view

A type of view, included in every database, that presents the information held in the system tables in an easily understood format.

temporary table

A table that is created for the temporary storage of data. There are two types: global and local.

See also:

- “local temporary table” on page 122
- “global temporary table” on page 119

transaction

A sequence of SQL statements that comprise a logical unit of work. A transaction is processed in its entirety or not at all. SQL Anywhere supports transaction processing, with locking features built in to allow concurrent transactions to access the database without corrupting the data. Transactions end either with a COMMIT statement, which makes the changes to the data permanent, or a ROLLBACK statement, which undoes all the changes made during the transaction.
transaction log

A file storing all changes made to a database, in the order in which they are made. It improves performance and allows data recovery in the event the database file is damaged.

transaction log mirror

An optional identical copy of the transaction log file, maintained simultaneously. Every time a database change is written to the transaction log file, it is also written to the transaction log mirror file.

A mirror file should be kept on a separate device from the transaction log, so that if either device fails, the other copy of the log keeps the data safe for recovery.

See also: “transaction log” on page 135.

transactional integrity

In MobiLink, the guaranteed maintenance of transactions across the synchronization system. Either a complete transaction is synchronized, or no part of the transaction is synchronized.

transmission rule

In QAnywhere, logic that determines when message transmission is to occur, which messages to transmit, and when messages should be deleted.

trigger

A special form of stored procedure that is executed automatically when a user runs a query that modifies the data.

See also:
- “row-level trigger” on page 131
- “statement-level trigger” on page 133
- “integrity” on page 120

UltraLite

A database optimized for small, mobile, and embedded devices. Intended platforms include cell phones, pagers, and personal organizers.

UltraLite runtime

An in-process relational database management system that includes a built-in MobiLink synchronization client. The UltraLite runtime is included in the libraries used by each of the UltraLite programming interfaces, as well as in the UltraLite engine.

unique constraint

A restriction on a column or set of columns requiring that all non-null values are different. A table can have multiple unique constraints.
unload

Unloading a database exports the structure and/or data of the database to text files (SQL command files for the structure, and ASCII comma-separated files for the data). You unload a database with the Unload utility.

In addition, you can unload selected portions of your data using the UNLOAD statement.

upload

The stage in synchronization where data is transferred from a remote database to a consolidated database.

user-defined data type

See “domain” on page 116.

validate

To test for particular types of file corruption of a database, table, or index.

view

A SELECT statement that is stored in the database as an object. It allows users to see a subset of rows or columns from one or more tables. Each time a user uses a view of a particular table, or combination of tables, it is recomputed from the information stored in those tables. Views are useful for security purposes, and to tailor the appearance of database information to make data access straightforward.

window

A group of rows over which an analytic function is performed. A window may contain one, many, or all rows of data that has been partitioned according to the grouping specifications provided in the window definition. The window moves to include the number or range of rows needed to perform the calculations for the current row in the input. The main benefit of the window construct is that it allows additional opportunities for grouping and analysis of results, without having to perform additional queries.

Windows

The Microsoft Windows family of operating systems, such as Windows Vista, Windows XP, and Windows 200x.

Windows CE

See “Windows Mobile” on page 136.

Windows Mobile

A family of operating systems produced by Microsoft for mobile devices.
**work table**

An internal storage area for interim results during query optimization.
Index

Symbols

$adapters
  MobiLink Listener action variable, 44
$best_adapter_mac
  MobiLink Listener action variable, 44
$best_adapter_name
  MobiLink Listener action variable, 44
$best_ip
  MobiLink Listener action variable, 44
$best_network_name
  MobiLink Listener action variable, 44
$content
  MobiLink Listener action variable, 44
$day
  MobiLink Listener action variable, 44
$hour
  MobiLink Listener action variable, 44
$message
  MobiLink Listener action variable, 44
  MobiLink Palm Listener Configuration action variable, 51
$message_end
  MobiLink Listener action variable, 44
  MobiLink Palm Listener Configuration action variable, 51
$message_start
  MobiLink Listener action variable, 44
  MobiLink Palm Listener Configuration action variable, 51
$minute
  MobiLink Listener action variable, 44
$ml_connect
  MobiLink Listener action variable, 44
$ml_password
  MobiLink Listener action variable, 44
$ml_user
  MobiLink Listener action variable, 44
$month
  MobiLink Listener action variable, 44
$network_name
  MobiLink Listener action variable, 44
$priority
  MobiLink Listener action variable, 44
$remote_id
  MobiLink Listener action variable, 44
$request_id
  MobiLink Listener action variable, 44
$second
  MobiLink Listener action variable, 44
$sender
  MobiLink Listener action variable, 44
  MobiLink Palm Listener Configuration action variable, 51
$subject
  MobiLink Listener action variable, 44
$time
  MobiLink Palm Listener Configuration action variable, 51
$type
  MobiLink Listener action variable, 44
$year
  MobiLink Listener action variable, 44
-a option
  MobiLink [dbsn], 38
-d option
  MobiLink [dbsn], 38
-e option
  MobiLink [dbsn], 38
-f option
  MobiLink [dbsn], 38
-ga option
  MobiLink [dbsn], 38
-gi option
  MobiLink [dbsn], 38
-i option
  MobiLink [dbsn], 38
-l option
  MobiLink [dbsn], 41
  MobiLink [dbsncfg], 50
-m option
  MobiLink [dbsn], 38
-n option
  MobiLink [dbsncfg], 50
-ni option
  MobiLink [dbsn], 38
-np option
  MobiLink [dbsn], 38
-ns option
  MobiLink [dbsn], 38
-nu option
  MobiLink [dbsn], 38
-o option
MobiLink [dblsn], 38
-os option
MobiLink [dblsn], 38
-ot option
MobiLink [dblsn], 38
-p option
MobiLink [dblsn], 38
-pc option
MobiLink [dblsn], 38
-q option
MobiLink [dblsn], 38
-qa option
MobiLink [dblsn], 38
-r option
MobiLink [dblsn], 38
-t option
MobiLink [dblsn], 38
-u option
MobiLink [dblsn], 38
-v option
MobiLink [dblsn], 38
-w option
MobiLink [dblsn], 38
-x option
MobiLink [dblsn], 38
-y option
MobiLink [dblsn], 38
@data option
MobiLink Listener [dblsn], 33
@filename option
MobiLink Listener [dblsn], 33
_BEST_IP_CHANGED_
server-initiated synchronization, 32
_generic_
MobiLink server-initiated synchronization
network_provider_id, 74
_IP_CHANGED_
server-initiated synchronization, 32

A
a_palm_msg structure
Palm Listener SDK, 91
action
MobiLink [dblsn], 42
MobiLink [dblsncfg], 51
action variables
MobiLink [dblsn], 44
MobiLink [dblsncfg] for Palm, 51
agent IDs
glossary definition, 109
altaction
MobiLink [dblsn], 42
articles
glossary definition, 109
atomic transactions
glossary definition, 109

B
base tables
glossary definition, 109
begin_connection property
Notifier, 56
begin_poll
using to create push requests, 10
begin_poll property
Notifier, 59
bit arrays
glossary definition, 109
bugs
providing feedback, xi
business rules
glossary definition, 110

carrier gateway
Notifier properties, 74
Carrier gateway properties
enable, 74
network_provider_id, 74
sms_email_domain, 75
sms_email_user_prefix, 75
carrier properties
server-initiated synchronization, 74
carriers
about, server-initiated synchronization, 21
configuring for server-initiated synchronization, 20
device tracking, 22
glossary definition, 110
properties, 74
server-initiated synchronization, 20
CE Phone Edition
setting up for server-initiated synchronization, 74
changes in connectivity
MobiLink [dblsn], 32
character sets
  glossary definition, 110
CHECK constraints
  glossary definition, 110
checkpoints
  glossary definition, 110
checksums
  glossary definition, 110
client event-hook procedures
  (see also event hooks)
client message store IDs
  glossary definition, 111
client message stores
  glossary definition, 110
client/server
  glossary definition, 111
code pages
  glossary definition, 111
collations
  glossary definition, 111
command files
  glossary definition, 111
common properties
  server-initiated synchronization, 54
communication streams
  glossary definition, 111
concurrency
  glossary definition, 111
config.notifier
  about, 15
configuring
  server-initiated synchronization, 14
configuring gateways and carriers
  server-initiated synchronization, 20
configuring Notifiers
  MobiLink server-initiated synchronization, 18
configuring the consolidated database
  server-initiated synchronization, 10
configuring the Notifier
  server-initiated synchronization, 14
confirm_action property
  SYNC gateway, 70
confirm_delivery
  MobiLink [dbsln], 42
confirm_delivery property
  device tracker gateway, 66
  SMTP gateway, 68
  SYNC gateway, 70
UDP gateway, 72
confirm_timeout property
  SMTP gateway, 68
  SYNC gateway, 71
  UDP gateway, 72
confirmation handling
  server-initiated synchronization, 56
confirmation_handler property
  Notifier, 56
conflict resolution
  glossary definition, 112
connect_string property
  Notifier, 63
connection IDs
  glossary definition, 112
connection profiles
  glossary definition, 112
connection-initiated synchronization
  MobiLink [dbsln], 32
connectivity changes
  MobiLink [dbsln], 32
consolidated databases
  glossary definition, 112
  server-initiated synchronization, 10
constraints
  glossary definition, 112
contention
  glossary definition, 113
continue
  MobiLink [dbsln], 42
correlation names
  glossary definition, 113
coverage-initiated synchronization
  MobiLink [dbsln], 32
creating push requests
  server-initiated synchronization, 12
creating the push request table
  server-initiated synchronization, 10
creator ID
  glossary definition, 113
cursor positions
  glossary definition, 113
cursor result sets
  glossary definition, 113
cursors
glossary definition, 113
deadlocks
glossary definition, 115
Default-DeviceTracker
server-initiated synchronization, 66
deleting push requests
server-initiated synchronization, 12
delivery confirmation
Notifier confirmation_handler property, 56
deploying
MobiLink server-initiated synchronization, 6
deployment considerations
server-initiated synchronization, 6
description property
SMTP gateway, 68
SYNC gateway, 71
UDP gateway, 72
developer community
newsgroups, xi
Device dependent functions
Listener SDK for Palm, 100
device tracker gateway
about device tracking, 22
about gateways, 20
device tracker gateway properties
confirm_delivery, 66
enable, 66
Notifier, 66
server-initiated synchronization, 66
smtp_gateway, 66
sync_gateway, 67
udp_gateway, 67
device tracking
glossary definition, 115
Listener options to enable, 23
Palm devices, 9.0.0 clients, 24
properties, 66
server-initiated synchronization, 22
setting up, 22
stopping, 24
troubleshooting, 26
DeviceTracker
about, 22
properties, 66
direct row handling
glossary definition, 116
DML
glossary definition, 114
direct row handling
glossary definition, 116
DML
glossary definition, 114
direct row handling
glossary definition, 116
DML
glossary definition, 114
direct row handling
glossary definition, 116
DML
glossary definition, 114
direct row handling
glossary definition, 116
DML
glossary definition, 114
direct row handling
glossary definition, 116
DML
glossary definition, 114
 documentation

DBA authority
glossary definition, 115
dbft hide utility
server-initiated synchronization, 33
dblsn
Listener utility for Windows, 28
syntax, 38
DBLSN FULL SHUTDOWN
MobiLink [dblsn], 44
dblsn.txt
MobiLink Listener default parameters, 33
dblsncfg
syntax, 50
DBMS
glossary definition, 114
dbspaces
glossary definition, 115
DDL
glossary definition, 113
Index
conventions, vii
SQL Anywhere, vi
SQL Anywhere 11.0.0, vi
domains
glossary definition, 116
downloads
glossary definition, 116
dynamic SQL
glossary definition, 116

E
EBFs
glossary definition, 116
embedded SQL
glossary definition, 116
enable property
Carrier gateway, 74
device tracker gateway, 66
Notifier, 63
SMTP gateway, 68
SYNC gateway, 71
UDP gateway, 72
encoding
glossary definition, 116
end_connection property
Notifier, 59
end_poll property
Notifier, 60
error handling
server-initiated synchronization, 60
error_handler property
Notifier, 60
event model
glossary definition, 117
external logins
glossary definition, 117
extraction
glossary definition, 117

F
failover
glossary definition, 117
feedback
documentation, xi
providing, xi
reporting an error, xi
requesting an update, xi
FILE
glossary definition, 117
FILE message type
glossary definition, 117
file-based downloads
glossary definition, 117
file-definition database
glossary definition, 117
filter-action pairs
MobiLink [dbsln], 41
filtering by remote ID
server-initiated synchronization, 30
filters
MobiLink [dbsln], 42
MobiLink [dbsncfg], 51
finding out more and requesting technical assistance
technical support, xi
foreign key constraints
glossary definition, 118
foreign keys
glossary definition, 117
foreign tables
glossary definition, 118
full backups
glossary definition, 118

gateways
configuring for server-initiated synchronization, 20
device tracker, 66
device tracking, 22
glossary definition, 118
server-initiated synchronization, 20
SMTP properties for server-initiated synchronization, 67
SYNC properties for server-initiated synchronization, 70
troubleshooting, 26
UDP properties for server-initiated synchronization, 70, 71
gateways and carriers
server-initiated synchronization, 20
generated join conditions
glossary definition, 118
generation numbers
glossary definition, 118
getting help
technical support, xi  

global temporary tables  
glossary definition, 119

glossary  
list of SQL Anywhere terminology, 109

grant options  
glossary definition, 119

gui property  
Notifier, 64

H  
hash  
glossary definition, 119
help  
technical support, xi

histograms  
glossary definition, 119

hooks  
(see also event hooks)

I  
iAnywhere developer community  
newsgroups, xi

iAnywhere JDBC driver  
glossary definition, 119
icons  
used in this Help, x

identifiers  
glossary definition, 119
incremental backups  
glossary definition, 119
indexes  
glossary definition, 120

InfoMaker  
glossary definition, 120
inner joins  
glossary definition, 120
install-dir  
documentation usage, ix

integrated logins  
glossary definition, 120
integrity  
glossary definition, 120

Interactive SQL  
glossary definition, 120
isolation levels  
glossary definition, 120

isolation property  
Notifier, 64

J  
JAR files  
glossary definition, 120
Java classes  
glossary definition, 121
jConnect  
glossary definition, 121

JDBC  
glossary definition, 121
join conditions  
glossary definition, 121
join types  
glossary definition, 121
joins  
glossary definition, 121

K  
key joins  
glossary definition, 118

L  
libraries  
MobiLink listening libraries, 46

Listener options for device tracking  
server-initiated synchronization, 23

Listener SDK for Palm  
server-initiated synchronization, 89

Listener software development kit  
about, 90

Listener utility  
about, 28
syntax, 38

listener_port property  
UDP gateway, 73

Listeners  
architecture, 2
configuring and starting, 28
default parameters file, 33
device tracking options, 23
glossary definition, 121
limitations of UDP Listeners, 6
limitations on CE or PCs, 6
Palm devices, 52
SDK for Palm, 90
Windows [dbsn], 28
Listeners for Palm devices
  server-initiated synchronization, 49
listeners_are_9000 property
  SMTP gateway, 68
  UDP gateway, 72
listening libraries
  server-initiated synchronization, 46
local temporary tables
  glossary definition, 122
locks
  glossary definition, 122
log files
  glossary definition, 122
logical indexes
  glossary definition, 122
lsn_udp.dll
  server-initiated synchronization, 47
LsnMain function
  Listener SDK for Palm, 98
LTM
  glossary definition, 122

M
maintenance releases
  glossary definition, 123
materialized views
  glossary definition, 123
maydial
  MobiLink [dbsn], 42
message
  MobiLink [dbsn], 31
  MobiLink Palm Listener Configuration action
    variable, 51
message handlers
  MobiLink [dbsn] syntax, 41
  server-initiated synchronization, 29
    using the filters message, message_start, and sender, 31
message log
  glossary definition, 123
message processing interface
  Listener SDK for Palm, 91
message stores
  glossary definition, 123
message systems
  glossary definition, 123
message types
  glossary definition, 123
message_end
  MobiLink Palm Listener Configuration action
    variable, 51
message_start
  MobiLink [dbsn], 31
  MobiLink Palm Listener Configuration action
    variable, 51
metadata
  glossary definition, 124
mirror logs
  glossary definition, 124
ml_del_dev_addr procedure
  syntax, 79, 81
ml_del_list procedure
  syntax, 79, 82
ml_delete_device system procedure
  syntax, 80
ml_delete_device_address system procedure
  syntax, 81
ml_delete_listening system procedure
  syntax, 82
ml_set_dev_addr procedure
  syntax, 79, 85
ml_set_device system procedure
  syntax, 83
ml_set_device_address system procedure
  syntax, 85
ml_set_listening system procedure
  syntax, 87
mlsrv11
  -notifier option, 17
MobiLink
  glossary definition, 124
  server-initiated synchronization, 1
MobiLink clients
  glossary definition, 124
MobiLink Listener SDK
  about, 89
MobiLink Monitor
  glossary definition, 124
MobiLink server
  glossary definition, 124
MobiLink server farm
  Notifiers, 18
MobiLink synchronization
  server-initiated synchronization, 1
MobiLink system tables
glossary definition, 124
MobiLink users
glossary definition, 124
multi-channel listening
server-initiated synchronization, 33

N
natural joins
glossary definition, 118
network protocols
glossary definition, 124
network server
glossary definition, 125
network_provider_id property
Carrier gateway, 74
newsgroups
technical support, xi
normalization
glossary definition, 125
Notifier behavior properties
about, 55
Notifier events
about, 55
Notifier polling events
about, 55
Notifier properties
begin_connection, 56
begin_poll, 59
confirmation_handler, 56
connect_string, 63
enable, 63
end_connection, 59
end_poll, 60
error_handler, 60
gui, 64
isolation, 64
poll_every, 64
request_cursor, 62
request_delete, 62
server-initiated synchronization, 55
shared_database_connection, 65
shutdown_query, 63
Notifier properties file
about, 15
Notifiers
about, 17
database sync architecture, 2
configuring, 18
carrier gateway configuration, 20
glossary definition, 125
MobiLink server farm, 18
request_cursor property, 62
starting, 17
Notifiers in a server farm
MobiLink server-initiated synchronization, 18
notifying the Listener with sa_send_udp
about, 13

O
object trees
glossary definition, 125
ODBC
glossary definition, 125
ODBC Administrator
glossary definition, 125
ODBC data sources
glossary definition, 125
online books
PDF, vi
outer joins
glossary definition, 125

P
packages
glossary definition, 126
Palm Computing Platform
MobiLink listeners for Palm devices, 49
Palm devices
device tracking for, 24
Listener, 52
Palm Listener Configuration utility
syntax, 50
Palm Listener utilities
server-initiated synchronization, 50
palm_lsn_ret enumeration
Listener SDK for Palm, 97
PalmLsn.h
server-initiated synchronization, 90
PalmLsn.lib
server-initiated synchronization, 90
PalmLsnAllocate function
Listener SDK for Palm, 91
PalmLsnCheckConfigDB function
Listener SDK for Palm, 96
PalmLsnDupMessage function
Listener SDK for Palm, 92
PalmLsnDupSender function
Listener SDK for Palm, 93
PalmLsnDupTime function
Listener SDK for Palm, 94
PalmLsnFree function
Listener SDK for Palm, 92
PalmLsnGetConfigFileName
Listener SDK for Palm, 101
PalmLsnNormalHandleEvent
Listener SDK for Palm, 102
PalmLsnNormalStart
Listener SDK for Palm, 102
PalmLsnNormalStop
Listener SDK for Palm, 102
PalmLsnProcess function
Listener SDK for Palm, 102
PalmLsnSpecialLaunch
Listener SDK for Palm, 95
PalmLsnTargetCompanyID
Listener SDK for Palm, 103
PalmLsnTargetDeviceID
Listener SDK for Palm, 101
parse trees
  glossary definition, 126
password property
  SMTP gateway, 69
PDB
  glossary definition, 126
PDF
  documentation, vi
  performance statistics
  glossary definition, 126
persistent connections
  dblsn -pc option, 38
personal server
  glossary definition, 126
physical indexes
  glossary definition, 126
plug-in modules
  glossary definition, 126
policies
  glossary definition, 126
poll_every property
  Notifier, 64
polling
  glossary definition, 126
post
  MobiLink [dblsn], 43
posting
  Windows messages to window classes in MobiLink, 43
PowerDesigner
  glossary definition, 126
PowerJ
  glossary definition, 127
predicates
  glossary definition, 127
primary key constraints
  glossary definition, 127
primary keys
  glossary definition, 127
primary tables
  glossary definition, 127
properties
  Notifier, 14
  server-initiated synchronization, 14
proxy tables
  glossary definition, 127
public wireless carriers
  configuration for server-initiated synchronization, 74
publication updates
  glossary definition, 128
publications
  glossary definition, 127
publisher
  glossary definition, 128
push notifications
  glossary definition, 128
push request table
  about, 10
push requests
  about, 10
  architecture, 2
  creating, 12
  creating the push request table, 10
  deleting, 12
  glossary definition, 128
  request_cursor property, 62
  sending, 12
push technology
  server-initiated synchronization, 1
Index

Q
QAnywhere
glossary definition, 128
QAnywhere Agent
glossary definition, 128
queries
glossary definition, 128
quick start
server-initiated synchronization, 7

R
RDBMS
glossary definition, 129
Redirector
glossary definition, 128
reference databases
glossary definition, 129
referenced object
glossary definition, 129
referencing object
glossary definition, 129
referential integrity
glossary definition, 129
regular expressions
glossary definition, 129
remote databases
glossary definition, 129
REMOTE DBA authority
glossary definition, 130
remote IDs
filtering for server-initiated synchronization, 30
glossary definition, 130
replication
glossary definition, 130
Replication Agent
glossary definition, 130
replication frequency
glossary definition, 130
replication messages
glossary definition, 130
Replication Server
glossary definition, 130
request_cursor property
Notifier, 62
request_delete property
Notifier, 62
role names
glossary definition, 131
roles
glossary definition, 130
rollback logs
glossary definition, 131
row-level triggers
glossary definition, 131
run
MobiLink [dbln], 42

S
sa_send_udp system procedure
using to notify a Listener, 13
samples-dir
documentation usage, ix
scheduling
MobiLink server-initiated synchronization, 44
schemas
glossary definition, 131
script versions
glossary definition, 131
script-based uploads
glossary definition, 131
scripts
glossary definition, 131
SDKs
Listener SDKs, 90
secured features
glossary definition, 131
sender
MobiLink [dbln], 31
MobiLink Palm Listener Configuration action
variable, 51
sender property
SMTP gateway, 69
UDP gateway, 73
sender_port property
UDP gateway, 73
sending push requests
server-initiated synchronization, 12
server initiated synchronization (see server-initiated synchronization)
server management requests
glossary definition, 132
server message stores
glossary definition, 132
server property
SMTP gateway, 69
server stored procedures
MobiLink server-initiated synchronization, 77
server-initiated synchronization
about, 1
architecture, 4
automatic connection recovery, 64
configuring and starting the Listener, 28
glossary definition, 132
Listener SDKs, 90
listening libraries, 46
Palm devices and 9.0.0 clients, 24
quick start, 7
shared database connections, 65
supported platforms, 5
system procedures, 77
unguaranteed delivery, 6
server-initiated synchronization system procedures
about, 77
services
glossary definition, 132
session-based synchronization
glossary definition, 132
setting properties
server-initiated synchronization, 14
setting properties in more than one place
server-initiated synchronization, 14
setting up device tracking
server-initiated synchronization, 22
setting up server-initiated synchronization
about, 9
setting up the Listener
server-initiated synchronization, 28
setting up the Notifier
server-initiated synchronization, 17
shared_database_connection property
Notifier, 65
shutdown_query property
Notifier, 65
sis (see server-initiated synchronization)
sms_email_domain property
Carrier gateway, 75
sms_email_user_prefix property
Carrier gateway, 75
SMTP gateway
about gateways, 20
listening libraries for server-initiated
cynchronization, 46
SMTP gateway properties
confirm_delivery, 68
confirm_timeout, 68
description, 68
enable, 68
listeners_are_900, 68
Notifier, 67
password, 69
sender, 69
server, 69
server-initiated synchronization, 67
user, 69
smtp_gateway property
device tracker gateway, 66
snapshot isolation
glossary definition, 132
socket
MobiLink [dblsn], 43
software development kits
MobiLink server-initiated synchronization, 90
SQL
glossary definition, 132
SQL Anywhere
documentation, vi
glossary definition, 132
SQL Remote
glossary definition, 133
SQL statements
glossary definition, 133
SQL-based synchronization
glossary definition, 132
start
MobiLink [dblsn], 42
starting the Notifier
server-initiated synchronization, 17
statement-level triggers
glossary definition, 133
stopping device tracking
server-initiated synchronization, 24
stored procedures
glossary definition, 133
string literal
glossary definition, 133
subqueries
glossary definition, 133
subscriptions
glossary definition, 133
support
newsgroups, xi
supported platforms
  server-initiated synchronization, 5
Sybase Central
  glossary definition, 134
SYNC gateway
  about gateways, 20
  Notifier properties, 70
SYNC gateway properties
  confirm_action, 70
  confirm_delivery, 70
  confirm_timeout, 71
description, 71
enable, 71
server-initiated synchronization, 70
sync_gateway property
  device tracker gateway, 67
synchronization
  glossary definition, 134
  server-initiated, 1
synchronization subscriptions
(see also subscriptions)
syntax
  MobiLink Listener [dblsn], 38
  MobiLink Palm Listener Configuration [dblsncfg], 50
  MobiLink server-initiated synchronization system procedures, 77
SYS
  glossary definition, 134
system objects
  glossary definition, 134
system procedures
  ml_del_dev_addr, 79, 81
  ml_del_listen, 79, 82
  ml_delete_device, 80
  ml_delete_device_address, 81
  ml_delete_listening, 82
  ml_set_dev_addr, 79, 85
  ml_set_device, 83
  ml_set_device_address, 85
  ml_set_listening, 87
  MobiLink server-initiated synchronization, 77
system tables
  glossary definition, 134
system views
  glossary definition, 134

T
technical support
  newsgroups, xi
template.notifier
    about, 15
temporary tables
  glossary definition, 134
time
  MobiLink Palm Listener Configuration action variable, 51
topics
  graphic icons, x
trapped address is not correct
  troubleshooting device tracking, 26
transaction log
  glossary definition, 135
transaction log mirror
  glossary definition, 135
transactional integrity
  glossary definition, 135
transactions
  glossary definition, 134
transmission rules
  glossary definition, 135
Treo
  Palm Listener utility, 52
Treo650.c
  server-initiated synchronization, 90
triggers
  glossary definition, 135
troubleshooting
  newsgroups, xi
  server-initiated synchronization gateways, 26

U
UDP gateway
  about gateways, 20
  listening libraries for server-initiated synchronization, 47
  Notifier properties, 70, 71
UDP gateway properties
  confirm_delivery, 72
  confirm_timeout, 72
description, 72
enable, 72
listener_port, 73
listeners_are_900, 72
sender, 73
sender_port, 73
server-initiated synchronization, 70, 71
udp_gateway property
device tracker gateway, 67
UltraLite
glossary definition, 135
UltraLite runtime
glossary definition, 135
unique constraints
glossary definition, 135
unload
glossary definition, 136
unreachable addresses
troubleshooting device tracking, 26
uploads
glossary definition, 136
user property
SMTP gateway, 69
user-defined data types
glossary definition, 136
using device tracking
Palm devices and 9.0.0 clients, 24
using subject and content filters
server-initiated synchronization, 30
using the filters message, message_start, and sender
server-initiated synchronization, 31
utilities
MobiLink Listener [dblsn], 38
MobiLink Palm Listener Configuration [dblsncfg], 50

V
validate
glossary definition, 136
variables
MobiLink [dblsn] action variables, 44
MobiLink [dblsncfg] Palm action variables, 51
verbosity
server-initiated synchronization, 54
verbosity property
Notifier, 54
views
glossary definition, 136

W
window (OLAP)