SYBASE iAnywhere

MobiLink™
Server-Initiated Synchronization

Contents

About this book

This book describes MobiLink server-initiated synchronization, a feature that allows the MobiLink server to initiate synchronization or perform actions on remote devices.

About the SQL Anywhere documentation

The complete SQL Anywhere documentation is available in four formats:

- **DocCommentXchange** DocCommentXchange is a community for accessing and discussing SQL Anywhere documentation on the web.

  To access the documentation, go to http://dcx.sybase.com.

- **HTML Help** On Windows platforms, the HTML Help contains the complete SQL Anywhere documentation, including the books and the context-sensitive help for SQL Anywhere tools.

  To access the documentation, choose **Start** » **Programs** » **SQL Anywhere 12** » **Documentation** » **HTML Help (English)**.

- **Eclipse** On Unix platforms, the complete Help is provided in Eclipse format. To access the documentation, run *sadoc* from the *bin32* or *bin64* directory of your SQL Anywhere 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 access the PDF documentation on Windows operating systems, choose **Start** » **Programs** » **SQL Anywhere 12** » **Documentation** » **PDF (English)**.

  To access the PDF documentation on Unix operating systems, use a web browser to open */documentation/en/pdf/index.html* under the SQL Anywhere installation directory.

Documentation conventions

This section lists the conventions used in this documentation.

Operating systems

SQL Anywhere runs on a variety of platforms. Typically, the behavior of the software is the same on all platforms, but there are variations or limitations. These are commonly based on the underlying operating system (Windows, Unix), and seldom on the particular variant (IBM AIX, Windows Mobile) or version.

To simplify references to operating systems, the documentation groups the supported operating systems as follows:

- **Windows** The Microsoft Windows family includes platforms that are used primarily on server, desktop, and laptop computers, as well as platforms used on mobile devices. Unless otherwise specified, when the documentation refers to Windows, it refers to all supported Windows-based platforms, including Windows Mobile.

  Windows Mobile is based on the Windows CE operating system, which is also used to build a variety of platforms other than Windows Mobile. Unless otherwise specified, when the documentation refers to Windows Mobile, it refers to all supported platforms built using Windows CE.

- **Unix** Unless otherwise specified, when the documentation refers to Unix, it refers to all supported Unix-based platforms, including Linux and Mac OS X.

For the complete list of platforms supported by SQL Anywhere, see "Supported platforms" [*SQL Anywhere 12 - Introduction*].

Directory and file names

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

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

- **Uppercase and lowercase directory names** On Windows and Unix, directory and file names may contain uppercase and lowercase letters. When directories and files are created, the file system preserves letter case.

  On Windows, references to directories and files are *not* case sensitive. Mixed case directory and file names are common, but it is common to refer to them using all lowercase letters. The SQL Anywhere installation contains directories such as *Bin32* and *Documentation*.

  On Unix, references to directories and files *are* case sensitive. Mixed case directory and file names are not common. Most use all lowercase letters. The SQL Anywhere installation contains directories such as *bin32* and *documentation*.

  The documentation uses the Windows forms of directory names. You can usually convert a mixed case directory name to lowercase for the equivalent directory name on Unix.

- **Slashes separating directory and file names** The documentation uses backslashes as the directory separator. For example, the PDF form of the documentation is found in *install-dir* *\Documentation\en\PDF* (Windows form).

  On Unix, replace the backslash with the forward slash. The PDF documentation is found in *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, executable file names have no suffix.

  For example, on Windows, the network database server is *dbsrv12.exe*. On Unix, it is *dbsrv12*.

- ***install-dir*** During the installation process, you choose where to install SQL Anywhere. The environment variable SQLANY12 is created and refers to this location. The documentation refers to this location as *install-dir*.

  For example, the documentation may refer to the file *install-dir/readme.txt*. On Windows, this is equivalent to *%SQLANY12%\readme.txt*. On Unix, this is equivalent to *$SQLANY12/readme.txt* or *${SQLANY12}/readme.txt*.

  For more information about the default location of *install-dir*, see "SQLANY12 environment variable" [*SQL Anywhere Server - Database Administration*].

- ***samples-dir*** During the installation process, you choose where to install the samples included with SQL Anywhere. The environment variable SQLANYSAMP12 is created and refers to this location. The documentation refers to this location as *samples-dir*.

  To open a Windows Explorer window in *samples-dir*, choose **Start** » **Programs** » **SQL Anywhere 12** » **Sample Applications And Projects**.

  For more information about the default location of *samples-dir*, see "SQLANYSAMP12 environment variable" [*SQL Anywhere Server - Database Administration*].

Command prompts and command shell syntax

Most operating systems provide one or more methods of entering commands and parameters using a command shell or command prompt. Windows command prompts include Command Prompt (DOS prompt) and 4NT. Unix command shells include Korn shell and bash. Each shell has features that extend its capabilities beyond simple commands. These features are driven by special characters. The special characters and features vary from one shell to another. Incorrect use of these special characters often results in syntax errors or unexpected behavior.

The documentation provides command line examples in a generic form. If these examples contain characters that the shell considers special, the command may require modification for the specific shell. The modifications are beyond the scope of this documentation, but generally, use quotes around the parameters containing those characters or use an escape character before the special characters.

These are some examples of command line syntax that may vary between platforms:

- **Parentheses and curly braces** Some command line options require a parameter that accepts detailed value specifications in a list. The list is usually enclosed with parentheses or curly braces. The documentation uses parentheses. For example:

  ```
  -x tcpip(host=127.0.0.1)
  ```

  Where parentheses cause syntax problems, substitute curly braces:

  ```
  -x tcpip{host=127.0.0.1}
  ```

  If both forms result in syntax problems, the entire parameter should be enclosed in quotes as required by the shell:

  ```
  -x "tcpip(host=127.0.0.1)"
  ```

- **Semicolons** On Unix, semicolons should be enclosed in quotes.
- **Quotes** If you must specify quotes in a parameter value, the quotes may conflict with the traditional use of quotes to enclose the parameter. For example, to specify an encryption key whose value contains double-quotes, you might have to enclose the key in quotes and then escape the embedded quote:

  ```
  -ek "my \"secret\" key"
  ```

  In many shells, the value of the key would be `my "secret" key`.
- **Environment variables** The documentation refers to setting environment variables. In Windows shells, environment variables are specified using the syntax *%ENVVAR%*. In Unix shells, environment variables are specified using the syntax *$ENVVAR* or *${ENVVAR}*.

Contacting the documentation team

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

You can leave comments directly on help topics using DocCommentXchange. DocCommentXchange (DCX) is a community for accessing and discussing SQL Anywhere documentation. Use DocCommentXchange to:

- View documentation
- Check for clarifications users have made to sections of documentation
- Provide suggestions and corrections to improve documentation for all users in future releases

Go to http://dcx.sybase.com.

Finding out more and requesting technical support

Newsgroups

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 details about your problem, including the build number of your version of SQL Anywhere. You can find this information by running the following command: **dbeng12 -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
- ianywhere.public.sqlanywhere.qanywhere

For web development issues, see http://groups.google.com/group/sql-anywhere-web-development.

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, and other staff, assist on the newsgroup service when they have time. They offer their help on a volunteer basis and may not be available regularly to provide solutions and information. Their ability to help is based on their workload.

Developer Centers

The **SQL Anywhere Tech Corner** gives developers easy access to product technical documentation. You can browse technical white papers, FAQs, tech notes, downloads, techcasts and more to find answers to your questions as well as solutions to many common issues. See http://www.sybase.com/developer/library/sql-anywhere-techcorner.

The following table contains a list of the developer centers available for use on the SQL Anywhere Tech Corner:

Name	URL	Description
SQL Anywhere .NET Developer Center	www.sybase.com/developer/library/sql-anywhere-techcorner/microsoft-net	Get started and get answers to specific questions regarding SQL Anywhere and .NET development.
PHP Developer Center	www.sybase.com/developer/library/sql-anywhere-techcorner/php	An introduction to using the PHP (PHP Hypertext Preprocessor) scripting language to query your SQL Anywhere database.

Name	URL	Description
SQL Anywhere Windows Mobile Developer Center	www.sybase.com/developer/library/sql-anywhere-techcorner/windows-mobile	Get started and get answers to specific questions regarding SQL Anywhere and Windows Mobile development.

Introduction to server-initiated synchronization

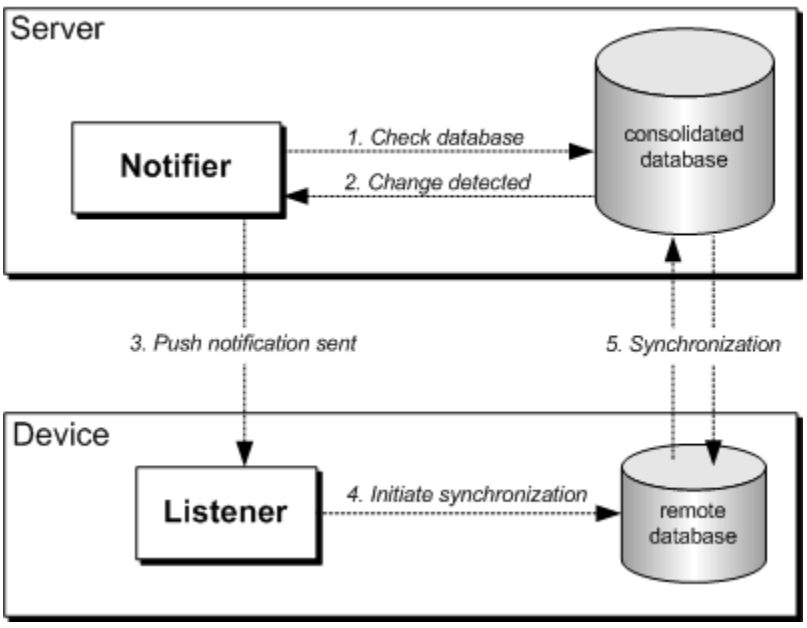
MobiLink server-initiated synchronization allows you to initiate synchronization from a consolidated database. You can send push notifications to remote databases, and cause remote databases to update the consolidated database. This MobiLink component provides programmable options for detecting changes in the consolidated database to initiate synchronization, selecting devices to send push notifications to, and determining how devices react to those push notifications.

Example

A trucking organization issues mobile devices to their drivers. Each device runs a database that contains routes and delivery locations. When a driver submits a notice of a traffic disruption, the report is sent to a consolidated database. A server-side MobiLink component called a Notifier detects the report and sends a push notification to other drivers whose routes are affected by the disruption. This push notification causes the remote databases to synchronize so that the drivers can use an alternate route.

The server-initiated synchronization process

In the following illustration, the Notifier checks a consolidated database for changes. The Notifier sends a push notification to a device, resulting in the remote database being synchronized with the consolidated database.



During the server-initiated synchronization process, the following steps occur:

1. Using a query based on business logic, the Notifier checks a consolidated database for changes that need to be synchronized with the remote database.

2. When a change is detected, the Notifier prepares a push notification to send to the device.
3. The Notifier sends the push notification. Push notifications can be sent through a Device Tracker, UDP, SMTP, or SYNC gateway.
4. The Listener compares the subject, content, or sender against a message filter.
5. If the filter conditions are met, an action is initiated. For example, in a typical implementation, an action could run the MobiLink client or launch an UltraLite application.

Server-initiated synchronization components

MobiLink server-initiated synchronization requires the following components:

- **Push requests** A push request is a row of values in a result set that tells a Notifier that you want to send a push notification to a device. Push requests cause server-initiated synchronizations to occur. Any database application can create push requests, including the Notifier. For example, a push request could be created using a database trigger that is activated when a price changes. See "Push requests" on page 5.
- **A MobiLink Notifier** A Notifier is a program integrated into the MobiLink server. It frequently checks the consolidated database for push requests. You can control how often the Notifier checks for push requests by specifying its properties. You must specify business logic to check for push requests, and to determine which devices should be notified. A push notification is sent to a device when the Notifier detects a push request. See "Notifiers" on page 10.
- **A MobiLink Listener** A Listener is a program that runs on a device. It receives push notifications from the Notifier, then uses a message handler to filter messages and initiate an action. In a typical application, actions are synchronization calls, but applications are capable of performing other actions. You can configure the MobiLink Listener to act differently on push notifications from selected server sources, or on push notifications that contain specific content.

  On Windows devices, the MobiLink Listener is an executable program that you configure with command line options. To receive push notifications, the device must be turned on and the MobiLink Listener must be running. See "MobiLink Listener utility for Windows devices (dblsn)" on page 49.
- **A light weight poller** A light weight poller is a device application that polls for push notifications at a specified time interval. Using a light weight poller is an alternative to setting up a gateway, and is recommended because it does not require a persistent connection to the server, and can help extend battery life.

  The MobiLink Listener is a light weight poller that you can configure using MobiLink Listener command line options. Alternatively, you can use the light weight polling API to create your own light weight poller.

See:

- "Setting up light weight polling options" on page 15
- "Light weight polling API" on page 71

- **A gateway (An alternative to a light weight poller)** A gateway provides a Notifier interface for sending push notifications to a device. Gateways are an alternative to light weight pollers. You can send messages using a device tracking gateway, a SYNC gateway, a UDP gateway, or an SMTP gateway. See "Gateways and carriers" on page 19.

Server-initiated synchronization deployment considerations

Consider the following issues before deploying server-initiated synchronization applications.

Device limitations when using UDP gateways

- The IP address on the device must be addressable directly from the MobiLink server.
- IP tracking for UDP notification does not work if the IP address on a Windows device is not addressable directly from the MobiLink server.

Device tracking limitations

Adaptive Server Anywhere 9.0.0 or earlier MobiLink Listeners do not support device tracking. To use device tracking with these MobiLink Listeners, you must set up device tracking manually. See "Adding support for device tracking" on page 20.

Supported device platforms

The MobiLink Listener is supported on Windows and Windows Mobile.

Quick start guide to server-initiated synchronization

Before completing this procedure, you must set up MobiLink for normal synchronization. See "MobiLink - Getting Started".

1. On the MobiLink server, prepare your consolidated database to store push requests. See "Push request requirements" on page 5.
2. On the MobiLink server, configure Notifier events to create and manage push requests. See "Configuring Notifier events and properties" on page 11.
3. On the device, set up a light weight poller. See "Light weight pollers" on page 18.

If you do not want to use a light weight poller, set up a supported gateway on the MobiLink server. When using an SMTP gateway, you also need to configure a carrier. See "Gateways and carriers" on page 19.

4. On the device, set up a MobiLink Listener to filter messages and perform actions. See "Message handlers" on page 12.

For instructions on how to set up server-initiated synchronization using Sybase Central, see "Setting up server-initiated synchronization in a synchronization model" [*MobiLink - Getting Started*].

Other resources

- Sample applications are installed to the *samples-dir**MobiLink*\ directory. (For more information about *samples-dir*, see "Samples directory" [*SQL Anywhere Server - Database Administration*].) All applications related to server-initiated synchronization are located in directories with the **SIS_** prefix.

Setting up server-initiated synchronization

The following sections describe how to set up server-initiated synchronization:

Push requests

A push request is a row of values in a result set that a Notifier checks to determine if push notifications need to be sent to a device. A Notifier places the push request inside a push notification then sends the push notification. In a typical server-initiated synchronization set up, a push request contains message content and target device information. Before a push notification can be sent, you need to configure a Notifier event so that the Notifier can detect the push request.

Push request requirements

The requirements for push requests are dependent on the method the MobiLink server uses to communicate with devices. All push requests require subject and content columns. If you are using light weight pollers to poll for push notifications, create a poll key column to identify them. Create gateway and address columns if you are using gateways to send push notifications.

You do not need to create push request columns if they already exist on your system. After you have satisfied the push request requirements, you are can work with them. See "Working with push requests" on page 7.

Push request requirements when using light weight pollers (recommended)

You must create the following columns when you are using light weight pollers to poll for push notifications:

Column	Type	Description
Poll key	VARCHAR	The key used to identify a light weight poller. Each light weight poller sends a unique key to identify itself on the MobiLink server.
Subject	VARCHAR	The subject line of the message.
Content	VARCHAR	The content of the message.

Push request requirements when using gateways

Unless otherwise specified, you must create the following columns when using gateways to send push notifications:

Column	Type	Description
Request ID	INTEGER	Optional. The unique ID of a push request. This column name is required for some Notifier events. See "Notifier events" on page 29.
Gateway	VARCHAR	The name of the gateway to which the message is sent.
Subject	VARCHAR	The subject line of the message.
Content	VARCHAR	The content of the message.
Address	VARCHAR	The destination address of a device.
Resend interval	VARCHAR	Optional. The time interval between message resends. The resend interval is useful when using a UDP gateway on an unreliable network. The Notifier assumes that all attributes associated with the push requests do not change; subsequent updates are ignored after the first poll of the request. The Notifier automatically adjusts the next polling interval if a push notification must be sent before the next polling time. You can stop a push request from being sent using synchronization logic in the request_cursor event. Delivery confirmation from the intended MobiLink Listener may stop a subsequent resend. See "request_cursor event" on page 33.
Time to live	VARCHAR	Optional. The time until the resend expires.

Example

The following example satisfies the push request requirements for using light weight polling by creating the necessary columns in a SQL Anywhere consolidated database table:

```
CREATE TABLE PushRequest (
    req_id INTEGER DEFAULT AUTOINCREMENT PRIMARY KEY,
    poll_key VARCHAR(128),
    subject VARCHAR(128),
    content VARCHAR(128)
)
```

You only need to create this table, or something like it, if the push request columns are not available elsewhere. These columns can exist across multiple tables, in existing tables, or in a view.

See also

- "Configuring Notifier events and properties" on page 11
- "request_cursor event" on page 33

Working with push requests

Generating push requests

Before push requests can be generated, your consolidated database must contain the push request columns required for server-initiated synchronization, and you must be able to obtain the values with a single database query. Push requests are generated automatically when you provide a database query in the request_cursor event that selects push request columns. For more information on push request requirements, see "Push request requirements" on page 5.

Example

The consolidated database contains a table named PushRequest, which contains push requests for a light weight poller. A remote device is identified on the MobiLink server as unique_device_ID. The server uses the following SQL script to request synchronization:

```
INSERT INTO PushRequest (poll_key, subject, content) VALUES
('unique_device_ID', 'synchronize', 'ASAP');
```

Using this script to insert values is not enough to generate a push request. The values must be selected using a database query in the request_cursor event. To generate a push request, the server uses the following request_cursor event script:

```
SELECT poll_key, subject, content FROM PushRequest;
```

The push request is generated when the unique_device_ID device polls the server for push notifications. The push notification subject is **synchronize**, and the content is **ASAP**.

Push request limitations

The following table lists the push request limitations for each column:

Column	Type	Limitation
Request ID	INTEGER	This value must be a unique primary key.
Poll key	VARCHAR	Only required when using light weight pollers. There are no limitations on the poll key.

Column	Type	Limitation
Gateway	VARCHAR	Only required when using gateways. This value must be set to the name of an enabled gateway. You specify your own custom gateway name, or choose one of the following pre-configured gateway names: • **Default-DeviceTracker** • **Default-SMTP** • **Default-SYNC** • **Default-UDP** See "Using gateways as an alternative to light weight pollers" on page 19.
Subject	VARCHAR	Avoid using non-alphanumeric characters when setting this value. Braces, chevrons, double quotations, parenthesis, single quotations, and square brackets are reserved for internal use, and should not be used in the subject column.
Content	VARCHAR	There are no limitations on the message content.
Address	VARCHAR	Only required when using gateways. For UDP gateways, this value should be an IP address or hostname. Port number suffixes are supported in the following formats: • *IP-address*:*port-number* • *hostname*:*port-number* For SMTP gateways, this value should be an email address. For SYNC and device tracking gateways, this value should be the recipient name defined with the MobiLink Listener -t+ option. See "-t dblsn option" on page 60.
Resend interval	VARCHAR	By default, this value is measured in minutes. You can specify **S**, **M**, and **H** for units of seconds, minutes, and hours, respectively. You can also combine units; for example, `1H 30M 10S` informs the Notifier to resend the messages every one hour, thirty minutes, and ten seconds. If this value is null or not specified, the default is to send exactly once, with no resend.

Column	Type	Limitation
Time to live	VARCHAR	By default, this value is measured in minutes. You can specify **S**, **M**, and **H** for units of seconds, minutes, and hours, respectively. You can also combine units; for example, `3H 30M 10S` informs the Notifier to stop resending messages three hours, thirty minutes, and ten seconds after the initial send. If this value is null or not specified, the default is to send exactly once, with no resend.

Detecting push requests and sending push notifications

A Notifier detects a push request by frequently firing the request_cursor event. By default, a script is not specified for this event; you must provide a request_cursor event script so that the Notifier can detect push requests. In a typical application, a request_cursor event script is a SELECT statement. See "request_cursor event" on page 33.

The following example uses the ml_add_property system procedure to create a request_cursor event script for a custom Notifier named Simple. The SELECT statement informs the Notifier to detect push requests from a table named PushRequest.

```
CALL ml_add_property('SIS', 'Notifier(Simple)', 'request_cursor',
    'SELECT poll_key, subject, content FROM PushRequest'
);
```

Note
You must select columns in the same order as they are specified in the push request. See "Push request requirements" on page 5.

For more information about setting up Notifier events, see "MobiLink server settings for server-initiated synchronization" on page 25.

Deleting push requests

The Notifier resends push notifications if the information about the notified device is never updated after being sent and satisfied according to your business rules. Once push requests are satisfied, you need to prevent the Notifier from detecting old push requests. You can delete the push requests using a synchronization script if the push notifications were sent for synchronization purposes.

You can use the request_delete event to delete push requests by their request ID, however, your push request must contain a request ID column, and you must enable delivery confirmation.

See:

- "Push request requirements" on page 5
- "request_delete event" on page 34
- "MobiLink server settings for server-initiated synchronization" on page 25

Notifiers

A Notifier is a program integrated into the MobiLink server that frequently checks the consolidated database for push requests. Once push requests are detected, it sends push notifications to devices. A Notifier also executes a series of events, allowing you to create scripts for monitoring data, managing push requests, handling delivery confirmations, and handling errors.

Notifiers start when you first load the MobiLink server. You can have more than one Notifier running within a single instance of the MobiLink server. For an example of how to use multiple Notifiers, see the sample application located in the *samples-dir\MobiLink\SIS_MultipleNotifier*. For more information about *samples-dir*, see "Samples directory" [*SQL Anywhere Server - Database Administration*].

If a Notifier loses the database connection, it attempts to recover the connection until it regains access. After recovery, the Notifier continues to operate with the same configuration settings.

Notifiers in a MobiLink server farm

In MobiLink 11.0 and earlier, server-initiated synchronization in a MobiLink server farm could cause redundant push notifications, leading to additional synchronizations and increased load on the consolidated database in a MobiLink server farm. A Notifier can now run on every MobiLink server in the farm; the Notifiers, together, ensure that there are no redundant push notifications to the same MobiLink Listener. The mlsrv12 -lsc server option is used to pass information to other servers when they want to connect to the local MobiLink server. See "-lsc mlsrv12 option" [*MobiLink - Server Administration*].

This feature makes one Notifier the primary, and all other Notifiers secondary. The primary Notifier controls push notifications, either directly or indirectly, via the secondaries. The secondary Notifiers also route MobiLink Listener information to the primary Notifier, so it knows where the MobiLink Listeners are and how to reach them.

If the MobiLink server running the primary Notifier fails, the server farm chooses a new primary Notifier, and notifications continue.

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

To use this feature, the following mlsrv12 command line options are required on all MobiLink servers in the farm:

- "-lsc mlsrv12 option" [*MobiLink - Server Administration*]
- "-notifier mlsrv12 option" [*MobiLink - Server Administration*]
- "-zs mlsrv12 option" [*MobiLink - Server Administration*]

Example

On host001:

```
mlsrv12 -notifier -zs ml001 -lsc tcpip(host=host001;port=2439) ...
```


On host007:

```
mlsrv12 -notifier -zs ml007 -lsc tcpip(host=host007;port=2439) ...
```

Configuring Notifier events and properties

Notifier events allow you to embed scripts that manage the overall server-initiated synchronization process. For more information about the Notifier event sequence and how they are fired, see "Notifier events" on page 29.

For example, you can configure Notifier events to perform the following tasks:

- Determine what, how, and to whom information is sent in a push request using the request_cursor event.
- Create push requests that respond to changes in the consolidated database using the begin_poll event. (Advanced usage)
- Delete push requests using the request_delete event. (If required)
- Track Notifier polls and clean up table data using the end_poll event. (Advanced usage)

Notifier properties are similar to events. While events manage the notification process, properties manage Notifier behavior. For example, Notifier properties determine how often the Notifier should poll the consolidated database, and if the Notifier should be enabled on startup. Notifier properties and events are configured as server-side settings. For more information, see "MobiLink server settings for server-initiated synchronization" on page 25.

Starting a Notifier

When the MobiLink server loads, it starts all enabled Notifiers. To disable a Notifier, you must set the enable Notifier property value to false. See "Notifier properties" on page 41.

To start your Notifiers, use one of the following methods:

- Configure your Notifiers and run mlsrv12 with the -notifier option specified.
- If your settings are stored in a Notifier configuration file, load the database at the command line by running mlsrv12, and specify the file using the -notifier option. For example, if you want to use a file called *myfirst.Notifier*, the following command configures the MobiLink server to use the properties and events specified in the file:

  ```
  mlsrv12 ... -notifier c:\myfirst.Notifier
  ```

  If you are using QAnywhere, load the database at the command line by running mlsrv12 with the -m option specified.

See also

- "-notifier mlsrv12 option" [*MobiLink - Server Administration*]
- "MobiLink server settings for server-initiated synchronization" on page 25
- "Configuring Notifier events and properties" on page 11
- "-m mlsrv12 option" [*MobiLink - Server Administration*]

Listeners

A Listener is a program that runs on a device. It receives push notifications from a Notifier and initiates actions. Listeners can upload device tracking information to the consolidated database when using a gateway for server-initiated synchronization.

See also

- "Device tracking gateways" on page 20
- "Message handlers" on page 12
- "MobiLink Listener utility for Windows devices (dblsn)" on page 49

Example

The following command starts the MobiLink Listener utility for Windows devices:

```
dblsn -v2 -m -ot dblsn.log
    -l "poll_connect='host=localhost';
    poll_key=sis_user1;
    poll_every=10;
    subject=sync;
    action='start dbmlsync.exe
        -c server=rem1;uid=DBA;pwd=sql
        -ot dbmlsyncOut.txt -qc';"
```

This command loads a MobiLink Listener with verbosity set to level 2, enables message logging, and specifies that the server is located at **localhost**. The *dblsn.log* file is truncated before output is written to it. The MobiLink Listener polls for push notifications every 10 seconds. If the MobiLink Listener receives a push notification where the subject is named **sync**, the MobiLink client application is launched.

For more information about MobiLink Listener command line options for Windows, see "MobiLink Listener options for Windows devices" on page 50.

Message handlers

A message handler is a MobiLink Listener component that scans the message contents of a push notification to initiate an action. It can also be used to specify light weight polling options, such as the server location and the polling frequency.

Message handlers consist of the following components:

- **Filter keywords** After a push notification is pre-processed, you can use filter keywords to scan the message contents. When a filter condition is met, an action is initiated. For example, you can specify

the **subject** keyword to filter a message that contains a specific subject, or you can specify the **sender** keyword to filter messages received from a specific MobiLink server.

- **An action** An action is initiated after filter conditions are met on a message. In a typical application, you specify an action to initiate synchronization, but you can also perform other operations. To assist with error processing, you can specify an alternative action to handle instances when the original action fails.

- **Poll settings** Poll settings allow you to configure how the MobiLink Listener polls the MobiLink server for push notifications.

- **Options** Options allow you to control remote settings, such as delivery and action confirmation.

You can create a message handler with the dblsn -l option. Multiple message handlers can be specified.

See also

- "-l dblsn option" on page 55
- "MobiLink Listener keywords for Windows devices" on page 62
- "MobiLink Listener action commands for Windows devices" on page 65

Working with message handlers

When a push notification is received by a MobiLink Listener, it extracts a message, which is split up and divided into several keywords. The **message** keyword contains the entire message in a raw format. The message is then divided into **subject**, **content**, and **sender** keywords. These keywords are run through your message filter to determine which actions to initiate. For more information about using these keywords for filtering messages, see "Filtering a message" on page 13.

Filtering a message

A filter keyword is used to compare part of a push notification to a user-defined phrase. If the two phrases are textually equivalent, then an action is initiated. For more information about pre-processing push notifications for message filtering, see "Message syntax" on page 83.

Filter keywords can be specified by running the MobiLink Listener with the following syntax:

```
dblsn ... -l "filter-keyword-name='content to filter';action='...'"
```

You can use the -l option multiple times to create multiple filters, but you must also specify an action for every -l instance. Actions are only initiated when all filters are satisfied.

Each of the following keywords can only appear once in a message handler:

- **content** This and the subject keyword are recommended for filtering messages. Use this keyword to filter messages based on their content. For example:

  ```
  dblsn -l "content='your content filter here';action='...'"
  ```

- **subject** This and the content keyword are recommended for filtering messages. Use this keyword to filter messages based on their subject. For example:

  ```
  dblsn -l "subject='your subject filter here';action='...'"
  ```

- **message** Use this keyword to filter messages based on their raw data. Your filter value must match the exact length of the message. This keyword is not recommended since it has a variable structure. For more information about pre-processing push notifications for message filtering, see "Message syntax" on page 83.

- **message_start** Use this keyword to filter messages based on part of their raw data, starting from the beginning. For more information about pre-processing push notifications for message filtering, see "Message syntax" on page 83.

  When you specify this keyword, the MobiLink Listener creates the $message_start and $message_end action variables.

- **sender** Use this keyword to filter messages based on their sender. This keyword is useful for tracking push notifications sent by a particular Notifier. The value is dependent on the gateway being used. For UDP gateways, it is the IP address of the host of the gateway. For SYNC gateways, it is **MobiLink**. For SMTP gateways, it depends on your wireless carrier. See "Gateways and carriers" on page 19.

See also

- "MobiLink Listener keywords for Windows devices" on page 62
- "Action variables" on page 15

Initiating actions

When a message matches the conditions of a filter, an action is initiated. For more information about filtering push notifications, see "Filtering a message" on page 13.

Actions can be specified by running the MobiLink Listener with the following syntax:

```
dblsn ... -l "...;action='action-command command statement'"
```

The following action commands allow you to perform different tasks when a message is filtered:

- **START** Start an application, and allow it to run in the background.
- **RUN** Run an application and wait for it to complete before receiving more push notifications.
- **POST** Post a window message to a process that is already running. This command can only be used on Windows devices.
- **SOCKET** Send a message to an application using a TCP/IP connection.
- **DBLSN FULL SHUTDOWN** Shut down the MobiLink Listener.

For a detailed list of action commands and syntax reference information, see "MobiLink Listener action commands for Windows devices" on page 65.

For more information about incorporating action variables into an action, see "Action variables" on page 15.

Action variables

Action variables allow you to reference parts of a push notification from a message filter or an action. See "Initiating actions" on page 14 and "Filtering a message" on page 13.

How action variables are set

Most action variables are set automatically every time a push notification is received. The variable names are similar to the names specified in the message syntax. For example, *message* sets the $message action variable, while *subject* sets $subject, *sender* sets $sender, and *content* sets $content. See "Message syntax" on page 83.

Using action variables

Action variables are used in the command line when you run the MobiLink Listener. How they are used is dependent on the message handler, and the action you want to initiate. The following example demonstrates the use of the RUN action command, which is used to initiate the MobiLink client application:

```
dblsn ... -l "subject=publish;action='RUN dbmlsync.exe @dbmlsync.txt -n
$content'"
```

This message handler filters messages where the subject is textually equivalent to "publish". Once filtered, dbmlsync is run with the -n option, passing the $content action variable as a parameter. Assuming that *content* references the name of a synchronization publication, dbmlsync uses the publication to synchronize the device database with the consolidated database.

The following example demonstrates the use of an action variable to filter a message:

```
dblsn ... -l "subject=$content;action='RUN script.bat"
```

When **subject** is textually equivalent to **content**, this message handler filters messages. Once filtered, the device runs a custom batch script.

See also

- "MobiLink Listener action commands for Windows devices" on page 65
- "MobiLink Listener action variables for Windows devices" on page 68
- "Filtering messages by remote ID" on page 16

Setting up light weight polling options

You can use message handlers to handle polling operations. The light weight polling options allow you to specify the location of the server, the Notifier name, the polling frequency, and a poll key. Alternatively, you can use the light weight polling API to specify these properties.

Light weight polling options can be specified by running the MobiLink Listener with the following syntax:

```
dblsn ... -l
    "poll_connect=protocol-options;
    poll_notifier=Notifier-name;
    poll_key=identifier-string;
    poll_every=number-of-seconds;..."
```

A single message handler can only contain one of each of the following options:

- **poll_connect** Use this option to specify the protocol options required to connect to the server. Alternatively, you can use the dblsn -x option to specify the default protocol options. The poll_connect option overrides the default protocol options for the message handler.
- **poll_notifier** Use this option to specify the Notifier used by the MobiLink server to handle push requests. This option is required since the MobiLink server can host multiple Notifiers.
- **poll_key** Use this option to identify the MobiLink Listener to the Notifier. The MobiLink server uses this value to send push notifications intended for the device. In a typical application, this value should be the remote ID of the device.
- **poll_every** Use this option to specify how often the MobiLink Listener should poll the Notifier. By default, the MobiLink Listener automatically retrieves this value from the MobiLink server. This value is measured in seconds.

See also

- "Push request requirements" on page 5
- "Configuring Notifier events and properties" on page 11
- "Light weight pollers" on page 18
- "MobiLink Listener keywords for Windows devices" on page 62
- "Light weight polling API" on page 71

Advanced message handler features

Filtering messages by remote ID

You can filter messages by remote ID using the dblsn -r option, and the $remote_id action variable.

When synchronizing a SQL Anywhere remote database for the first time, a remote ID file containing the ID of your database is created. The file name is the same as the database, but has the *.rid* extension, and is stored in the same directory as the database. For UltraLite databases, there is no remote ID file; the remote ID is extracted from the database directly.

When you start the MobiLink Listener, use the dblsn -r option to provide the name and location of the remote ID file or the UltraLite database, then use the dblsn -l option to create your message handler.

You can type the remote ID directly into your message filter. However, remote IDs are GUID by default; the remote ID is not easy to remember unless you provide a meaningful name.

Note
In the dblsn command line, you can specify multiple instances of the -r and -l options. The $remote_id action variable used in a -l option is always specified in the -r option that precedes it. So, it is important to specify the -r option before the -l option.

The following example demonstrates the use of multiple remote IDs. It assumes that your device has 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.

```
dblsn ... -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

- "Action variables" on page 15
- "Remote IDs" [*MobiLink - Client Administration*]
- "-r dblsn option" on page 59
- "MobiLink Listener action variables for Windows devices" on page 68

Connectivity-initiated synchronization

On Windows devices, you can initiate synchronization when connectivity changes.

When IP connectivity is gained or lost, the device sends a push notification to the MobiLink Listener with the message **_IP_CHANGED_**. When the device finds a new optimum path to the MobiLink server, it sends a push notification to the MobiLink Listener with the message **_BEST_IP_CHANGED_**. Using a message handler, you can detect these changes in connectivity and initiate an action.

Identifying any change in connectivity

The **_IP_CHANGED_** message indicates that a change in IP connectivity has occurred. A change usually occurs when a device is within range of a WiFi network, when the user makes a RAS connection, or when the user puts the device in a cradle. You can reference the **_IP_CHANGED_** message by running the MobiLink Listener with the following syntax:

```
dblsn ... -l "message=_IP_CHANGED_;action='...'"
```

The following example demonstrates how to use the **_IP_CHANGED_** message. The message handler filters the message, and sends it to the server. If the connection is lost, an error is generated.

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

Identifying a change in the optimum path to a MobiLink server

The **_BEST_IP_CHANGED_** message indicates that a change in the optimum path to the MobiLink server has occurred. You can reference this message when you run the MobiLink Listener with the following syntax:

```
dblsn ... -x MobiLink-protocol-options -l
"message=_BEST_IP_CHANGED_;action='...'"
```

When filtering the **_BEST_IP_CHANGED_** message, the $best_ip action variable, which substitutes the local IP address that represents the best IP connection, can help you initiate useful actions. If there is no IP connection, $best_ip returns 0.0.0.0.

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

```
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 server=remote;uid=DBA;pwd=sql -n
test_pub'"
```

Note
When testing connectivity-initiated synchronization with your applications, run the MobiLink Listener on a separate computer from your MobiLink server.

See also

- “MobiLink Listener utility for Windows devices (dblsn)” on page 49
- “MobiLink Listener action commands for Windows devices” on page 65
- “MobiLink Listener action variables for Windows devices” on page 68

Light weight pollers

A light weight poller is a device application that polls for push notifications at a specified time interval. Using a light weight poller is an alternative to setting up a gateway, and is recommended because it does not require a persistent connection to the server like the SYNC gateway does, nor does it require a continuous connection like the UDP gateway does.

When a device polls the server, it sends a poll key and a Notifier name. The MobiLink server checks the Notifier name to determine which Notifier should check the cache for push requests. The poll key identifies the device to the Notifier, which uses the poll key to detect push requests intended for the device. Push notifications are sent after the push requests are detected.

Use MobiLink Listener command line options to configure a light weight poller. Alternatively, use the light weight polling API to integrate a light weight poller into your device application.

See also

- "Light weight polling API" on page 71
- "Setting up light weight polling options" on page 15
- "MobiLink Listener keywords for Windows devices" on page 62

Gateways and carriers

Using gateways as an alternative to light weight pollers

A gateway is 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. They are an alternative to light weight pollers and require a constant network connection.

Gateway properties are configured on a MobiLink server. You can configure multiple gateways on a single MobiLink server. See "MobiLink server settings for server-initiated synchronization" on page 25.

Supported gateways

The following gateways are supported on the MobiLink server:

- **SYNC gateway** The SYNC gateway is a TCP/IP-based gateway; push notifications are sent through the same protocol as your MobiLink synchronizations.

  The default SYNC gateway is named Default-SYNC. Typically, the default gateway settings do not need to be changed.

  See "SYNC gateway properties" on page 44.

- **UDP gateway** The UDP gateway sends push notifications through a UDP gateway.

  The default UDP gateway is named Default-UDP. Typically, the default gateway settings do not need to be changed. MobiLink Listeners use UDP by default when listening for push notifications.

  See "UDP gateway properties" on page 45.

- **SMTP gateway** The SMTP gateway sends push notifications using an email-to-SMS carrier service.

  The default SMTP gateway is named Default-SMTP.

  See "SMTP gateway properties" on page 43.

Device tracking gateway

In addition to the supported gateways, you can configure a device tracking gateway which automatically chooses the most appropriate gateway to send push notifications. The default device tracking gateway is Default-DeviceTracker. It is recommended that you use this gateway if you do not want to use a light weight poller. For more information about device tracking, see "Device tracking gateways" on page 20.

Device tracking gateways

Device tracking allows a MobiLink server to track devices using the remote ID information of a push request. A device tracking gateway uses automatically-tracked IP addresses, phone numbers, and public wireless network provider IDs to deliver push notifications through SYNC, UDP, and SMTP gateways. The gateway attempts to connect to the device using a SYNC gateway first. If delivery fails, a UDP gateway is attempted, followed by an SMTP gateway. This feature is useful when you expect device addresses to change.

A device tracking gateway can have a maximum of three subordinate gateways: one SYNC, one SMTP, and one UDP. Push notifications are automatically routed to one of the subordinate gateways based on the device tracking information sent by a MobiLink Listener. By enabling these subordinate gateways, device address changes are automatically managed by the MobiLink server. When an address changes, the MobiLink Listener synchronizes with the consolidated database to update the tracking information, which is located in the ml_device_address system table.

Most 9.0.1 or later MobiLink Listeners support device tracking. If you are using a MobiLink Listener that does not support device tracking, you can use a device tracking gateway by providing the tracking information yourself. See “Adding support for device tracking” on page 20.

See also

- “Using gateways as an alternative to light weight pollers” on page 19
- “Carriers and carrier configuration” on page 23

Adding support for device tracking

Note
You only need to support device tracking if you are using MobiLink Listeners running on Adaptive Server Anywhere 9.0.0 or earlier. Device tracking is already supported on all other MobiLink Listeners for Windows devices.

Several system procedures are available to manually set up device tracking for 9.0.0 MobiLink Listeners. These procedures update the ml_device, ml_device_address, and ml_listening MobiLink system tables on the consolidated database.

With manual device tracking, you can address recipients by MobiLink user name without providing network address information. However, the information cannot be automatically updated by MobiLink if it changes; you must change it manually. This method is especially useful for SMTP gateways because email addresses seldom change.

For UDP gateways, you can not rely on static entries if your IP address changes every time you reconnect. You can resolve this problem by addressing the host name instead of IP address. However, this solution slows updates to DNS server tables, and can misdirect push notifications. You can also set up system procedures to update the system tables programmatically.

To manually set up device tracking for 9.0.0 MobiLink Listeners

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

```
CALL ml_set_device(
    'myWindowsMobile',
    'MobiLink Listeners for myWindowsMobile - 9.0.1',
    '1',
    'not used',
    'y',
    'manually entered by administrator'
);
```

   The first parameter, **myWindowsMobile**, is a unique user-defined device name. The second parameter contains optional remarks about the MobiLink Listener version. The third parameter specifies a MobiLink Listener version; use **0** for SQL Anywhere 9.0.0 MobiLink Listeners, or **2** for post-9.0.0 MobiLink Listeners for Windows. The fourth parameter specifies optional device information. The fifth parameter specifies whether device tracking should be ignored. The final parameter contains optional remarks for this entry.

   See "ml_set_device system procedure" on page 78.

2. For each device, add an address record to the ml_device_address system table. For example:

```
CALL ml_set_device_address(
    'myWindowsMobile',
    'ROGERS AT&T',
    '55511234567',
    'y',
    'y',
    'manually entered by administrator'
);
```

   The first parameter, **myWindowsMobile**, is a user-defined unique device name. The second parameter is a network provider ID, which must match the network_provider_id carrier property. The third parameter is an IP address for UDP. The fourth parameter determines whether to activate this entry for sending push notifications. The fifth parameter specifies whether device tracking should be ignored. The final parameter contains optional remarks for this entry.

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

```
CALL ml_set_listening(
    'myULDB',
    'myWindowsMobile',
    '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 determines whether to activate this entry for device tracking addressing. The fourth parameter specifies whether device tracking should be ignored. The final parameter contains optional remarks for this entry.

See also

- "ml_set_listening system procedure" on page 80
- "ml_set_device_address system procedure" on page 79
- "Device tracking gateways" on page 20
- "Carrier properties" on page 46

Quick guide to device tracking gateway configuration

To set up device tracking

1. Set up a SYNC, UDP, or SMTP gateway.

   When you start the MobiLink server, these gateways are already set up with the default settings. For more information about configuring properties or creating your own gateways, see "MobiLink server settings for server-initiated synchronization" on page 25.

   Note
   The SMTP gateway requires carrier configuration. See "Carriers and carrier configuration" on page 23.

2. Create a new Notifier and set up your request_cursor event with the following conditions:
   - The gateway name must be set to the name of a device tracking gateway you want to use. The default gateway is named Default-DeviceTracker. This name is represented by the first column of the result set.
   - The address name must be set to the remote ID of the device. Use the dblsn -t+ option to register the remote ID with the MobiLink server. This name is represented by the fourth column of the result set.

   For more information about setting up a request_cursor event, see "request_cursor event" on page 33.

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

   The default MobiLink Listener name is *device_name*-**dblsn**, where *device_name* is the name of your device.

   Run the MobiLink Listener to view the device name, which can be found in the MobiLink Listener message window. Alternatively, you can set the device name using the dblsn -e option, or set a different MobiLink Listener name using the dblsn -u option. See "-e dblsn option" on page 54 and "-u dblsn option" on page 60.

   For more information about registering MobiLink users, see "Creating and registering MobiLink users" [*MobiLink - Client Administration*].

4. Start a MobiLink Listener with the required options. For more information about starting a MobiLink Listener, see "MobiLink Listener utility for Windows devices (dblsn)" on page 49.

Carriers and carrier configuration

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

You must configure a wireless carrier to send push notifications through an SMTP gateway because the Notifier needs to construct valid email addresses. You must also configure a wireless carrier when using a device tracking gateway with a subordinate SMTP gateway enabled.

Carrier properties, such as the network provider ID and the SMS email prefix, are configured on a MobiLink server. To accommodate multiple carrier services, configure multiple carriers on the MobiLink server. See "MobiLink server settings for server-initiated synchronization" on page 25.

Sender syntax

When a push notification is received by a MobiLink Listener and pre-processed for message filtering, it is divided into several keywords. The **sender** keyword in a **message** is an email address, which is generated by the device and varies depending on the wireless carrier. For more information about pre-processing messages, see "Message syntax" on page 83.

The **sender** syntax is in the following format:

```
sender = sms_email_user_prefix phone-number@sms_email_domain
```

Note
There are no spaces between *sms_email_user_prefix* and *phone-number*.

The *sms_email_user_prefix* and *sms_email_domain* values are carrier properties that should be configured on the MobiLink server. The *phone-number* value is taken from the address column of the ml_device_address system table.

To determine the sender syntax, run the MobiLink Listener on a device that uses a carrier service. Enable message logging, and set verbosity to level 2 using the dblsn -m and -v options. Check the message log after loading the MobiLink Listener.

See also

- "MobiLink server settings for server-initiated synchronization" on page 25
- "Carrier properties" on page 46

MobiLink server settings for server-initiated synchronization

Server-side settings consist of Notifier properties, gateway properties, carrier properties, and Notifier events. To configure these settings, use one of the following methods:

- Sybase Central
- A Notifier configuration file
- The ml_add_property system procedure

The Sybase Central and ml_add_property system procedure methods add events and settings to the ml_property system table.

Note
Changes made to server-side settings do not take effect while the MobiLink server is running. To apply new settings, you must shut down and restart the MobiLink server.

If you have already configured server-side settings in the ml_property system table and want to use a Notifier configuration file, the system table settings are always loaded first, followed by the file settings. The Notifier configuration file overwrites existing server-side settings, but the changes are not permanently applied to the consolidated database.

Configuring server-side settings using the ml_add_property system procedure

Use the ml_add_property system procedure to configure the server-side settings of a SQL Anywhere consolidated database. You can set these properties and events using Interactive SQL.

Note
You must use ANSI standard when naming your notifiers, gateways, and carriers.

Common properties syntax

```
CALL ml_add_property('SIS', '', 'Property', Value);
```

Notifier properties and events syntax

```
CALL ml_add_property('SIS', 'Notifier(NotifierName)', 'Event-or-Property',
Value);
```

Gateway properties syntax

```
CALL ml_add_property('SIS', 'DeviceTracker(DeviceTrackerName)', 'Property',
Value);

CALL ml_add_property('SIS', 'SMTP(SMTPName)', 'Property', Value);
```

```
CALL ml_add_property('SIS', 'UDP(UDPName)', 'Property', Value);
CALL ml_add_property('SIS', 'SYNC(SYNCName)', 'Property', Value);
```

Carrier properties syntax

```
CALL ml_add_property('SIS', 'Carrier(CarrierName)', 'Property', Value);
```

For more information, see "ml_add_property system procedure" [*MobiLink - Server Administration*].

Configuring server-side settings using Sybase Central

Sybase Central provides a graphical user interface for modifying properties and events. You can use Sybase Central to configure multiple Notifiers, gateways, and carriers. By configuring your server-side settings though Sybase Central, you do not need to specify a Notifier configuration file at the command line when using the mlsrv12 -notifier option.

Note
You must use ANSI standard when naming your notifiers, gateways, and carriers.

To set up Notifiers, gateways, and carriers using Sybase Central

1. Use the MobiLink plug-in to create a MobiLink project for your consolidated database if you have not already created one.

   See "Creating a MobiLink project" [*MobiLink - Getting Started*].

2. From the **View** menu, choose **Folders**.

3. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Consolidated databases**, your consolidated database name, and then select **Notification**.

   In the right pane, all available Notifiers, gateways, and carriers appear.

4. Create new Notifiers, gateways, and carriers.
   - To create a new Notifier, click the **Notifiers** tab in the right pane, then choose **File** » **New** » **Notifier**.
   - To create a new gateway, click the **Gateways** tab in the right pane, then choose **File** » **New** » **Gateway**.
   - To create a new carrier, click the **Carriers** tab in the right pane, then choose **File** » **New** » **Carrier**.

5. Select a Notifier, gateway, or carrier to configure.
   - To set up Notifier properties or events, click the **Notifiers** tab in the right pane and choose the Notifier that you want to configure.

- To set up gateway properties, click the **Gateways** tab in the right pane and choose the gateway that you want to configure.
- To set up carrier properties, click the **Carriers** tab in the right pane and choose the carrier that you want to configure.

Choose **File** » **Properties**.

A window appears where you can adjust all settings applicable to the chosen Notifier, gateway, or carrier.

6. Click **OK**.

Importing server-side settings from a Notifier configuration file

Server-side settings can be imported from a Notifier configuration file into the ml_property_table.

To import server-side settings from a Notifier configuration file

1. Use the MobiLink plug-in to create a MobiLink project for your consolidated database if you have not already created one.

   See "Creating a MobiLink project" [*MobiLink - Getting Started*].

2. From the **View** menu, choose **Folders**.
3. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Consolidated databases**, your consolidated database name, and then select **Notification**.
4. Choose **File** » **Import Settings**, and follow the instructions in the wizard.

Exporting server-side settings to a Notifier configuration file

You can export the server-side settings from the ml_property table into a Notifier configuration file. By exporting the settings, you can create multiple versions of your server-side settings, and you can load a different version using the mlsrc12 -notifier option.

To export server-side settings to a Notifier configuration file

1. Use the MobiLink plug-in to create a MobiLink project for your consolidated database if you have not already created one.

   See "Creating a MobiLink project" [*MobiLink - Getting Started*].

2. From the **View** menu, choose **Folders**.
3. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Consolidated databases**, your consolidated database name, and then select **Notification**.
4. Choose **File** » **Import Settings**, and follow the instructions in the wizard.

See also

- "Setting up server-initiated synchronization in a synchronization model" [*MobiLink - Getting Started*]

Configuring server-side settings using the Notifier configuration file

Server-side settings can be stored in a Notifier configuration file. You can use this file to configure multiple Notifiers, gateways, and carriers.

Note
You must use ANSI standard when naming your notifiers, gateways, and carriers.

Creating and configuring a Notifier configuration file

A Notifier configuration file can be created using a text editor, or it can be generated from property and event settings exported from Sybase Central. See "Configuring server-side settings using Sybase Central" on page 26.

To view the layout of a typical Notifier configuration file, open the *samples-dir\MobiLink \template.Notifier* template file, where *samples-dir* is the location of your SQL Anywhere 12 samples directory. The template file provides examples for configuring server-side properties and events.

When you have configured the necessary settings, save the Notifier configuration file and load your server-side properties and events into the MobiLink server.

Common properties syntax

```
Property = Value
```

Notifier events syntax

```
Notifier(NotifierName).Event = \
# Replace this text with SQL script.          \
# Be sure to put a backslash (\) at           \
# the end of every line of code               \
# if your event requires multiple             \
# lines of text.
```

Notifier properties syntax

```
Notifier(NotifierName).Property = Value
```

Gateway properties syntax

```
# For Device tracking gateways:
DeviceTracker(DeviceTrackerName).Property = Value
# For SMTP gateways:
SMTP(SMTPName).Property = Value
# For SYNC gateways:
SYNC(SYNCName).Property = Value
# For UDP gateways:
UDP(UDPName).Property = Value
```

Carrier properties syntax

```
Carrier(CarrierName).Property = Value
```


Loading a Notifier configuration file

To load a Notifier configuration file into the MobiLink server, run mlsrv12 from the command line with the -notifier option specified. For example, to use the server-side settings defined in a CarDealer.Notifier configuration file, run the following command:

```
mlsrv12 ... -notifier "c:\CarDealer.Notifier"
```

If a file is not specified, the *config.Notifier* file is loaded by default.

For more information on the mlsrv12 -notifier option, see "-notifier mlsrv12 option" [*MobiLink - Server Administration*].

Note
If you want to use the default SYNC gateway, you cannot store server-side settings in a Notifier configuration file. You must store them in the ml_property system table using an alternative method. See "MobiLink server settings for server-initiated synchronization" on page 25.

Using escape sequences

The backslash (\ ) is the escape character. The following is a list of common escape sequences that you can use in a Notifier configuration file:

Escape sequence	Description
\b	Backspace
\t	Tab
\n	Linefeed
\r	Carriage return
\"	Double quote (")
\'	Single quote (')
\\	Backslash (\ )
\e	Escape

Unicode escape sequences are of the form \u*XXXX* while ASCII escape sequences are of the form \x*XX*, where each *X* represents a hexadecimal digit.

When editing a property or event that requires multiple lines of text, add a single backslash character (\ ) at the end of each line.

Notifier events

Events are fired whenever a Notifier polls a MobiLink Listener. When an event is fired, the SQL script associated with the event is executed. You can incorporate SQL script into any of the Notifier events listed in this section. Although scripting is optional, you must script the request_cursor polling event.

There are three classifications of Notifier events: polling events, connection events, and asynchronous events. Polling events are fired every time a Notifier checks the consolidated database, and include all the events that occur between a begin_poll event and an end_poll event. Connection events are fired during the Notifier database connection. Asynchronous events can be fired at any time during the synchronization process.

Unless otherwise specified, Notifier events can be configured using any of the recommended methods. For more information about configuring Notifier events, see "MobiLink server settings for server-initiated synchronization" on page 25.

When a MobiLink Listener polls the Notifier, these events are fired in the following order:

```
Fire begin_connection event
For each poll (
    Fire begin_poll event
        Fire shutdown_query event
        Fire request_cursor event
        For all requests expired before required confirmation (
            Fire error_handler event
        )
        Fire request_delete event
    Fire end_poll event
)
Fire end_connection event
```

Polling events

Polling events are a classification of Notifier events that are fired every time a Notifier checks the consolidated database. These events include all the events that occur between a begin_poll event and an end_poll event.

begin_poll event

This polling event accepts SQL script and is fired before the Notifier checks the consolidated database for push requests. The value is null by default, so this event is not fired.

See also

- "Push requests" on page 5
- "Notifier events" on page 29
- "MobiLink server settings for server-initiated synchronization" on page 25

Example

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

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

```
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 event

This polling event accepts SQL script and is fired after the Notifier checks the consolidated database for push requests. The value is null by default, so this event is not fired.

You can use this event to perform table cleanup or to log the results of a poll.

See also

- “Notifier events” on page 29
- “MobiLink server settings for server-initiated synchronization” on page 25

error_handler event

Configure this event to indicate when a transmission fails or was not confirmed. For example, when a transmission fails, you can use this event to insert a row in an audit table or to send a push notification.

The following table details the parameters that can be captured using the error_handler event:

Script parameter	Type	Description
request_option (out)	Integer	Controls what the Notifier does to the push request after the error handler returns. The output can be one of the following values: • **0**: Perform default action based on the error code and log the error. • **1**: Do nothing. • **2**: Execute the request_delete event. • **3**: Attempt to deliver to a secondary gateway.

Script parameter	Type	Description
error_code (in)	Integer	Use one of the following values for the error code: • **-1**: The request timed out with confirmation of success. • **-8**: An error occurred during delivery attempt.
request_id (in)	Integer	Identifies the request.
gateway (in)	Varchar	Specifies the gateway associated with the push request.
address (in)	Varchar	Specifies the address associated with the push request.
subject (in)	Varchar	Specifies the subject associated with the push request.
content (in)	Varchar	Specifies the content associated with the push request.

Note
This event requires the use of a system procedure. You can not configure this event directly using Sybase Central. See "MobiLink server settings for server-initiated synchronization" on page 25.

See also

- "Notifier events" on page 29
- "MobiLink server settings for server-initiated synchronization" on page 25

Example

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

```
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 this ml_add_property system procedure with a SQL Anywhere consolidated database, run the following command:

```
call ml_add_property(
    'SIS',
    'Notifier(myNotifier)',
    'error_handler',
    'call CustomErrorHandler(?, ?, ?, ?, ?, ?, ?)');
```

Alternatively, you can fire this event by adding the following line to a Notifier configuration file:

```
Notifier(myNotifier).error_handler = call
CustomErrorHandler(?, ?, ?, ?, ?, ?, ?)
```

Run the file using the mlsrv12 -notifier option. For more information on how to configure a Notifier configuration file, see “Configuring server-side settings using the Notifier configuration file” on page 28.

request_cursor event

This polling event accepts SQL script and is fired to detect push requests. You must configure this event.

Fetching push requests when using a light weight poller (recommended)

When this event contains up to three columns in a result set, the Notifier acknowledges that there is no persistent connection between the server and the device, and that a device must poll the Notifier before push notifications can be sent. The Notifier caches the result set before sending push notifications. The MobiLink server identifies the device by the poll key, which is sent by the device every time the device polls the Notifier.

The result set of this event must contain the following columns in the specified order:

- Poll key

- Subject (optional)
- Content (optional)

Fetching push requests when using a gateway

When this event contains more than three columns in a result set, the Notifier acknowledges that a persistent connection exists between the server and the device, and then sends push notifications using a gateway when push requests are detected.

The result set of this event must contain the following columns in the specified order:

- Request ID (optional)
- Gateway
- Subject
- Content
- Address
- Resend interval (optional)
- Time to live (optional)

See also

- “Push request requirements” on page 5
- “Notifier events” on page 29
- “MobiLink server settings for server-initiated synchronization” on page 25

Example

The following example uses the ml_add_property system procedure to create a request_cursor event script for a custom Notifier named Simple. The SELECT statement tells the Notifier to detect push requests from a table named PushRequest.

```
CALL ml_add_property('SIS', 'Notifier(Simple)', 'request_cursor',
    'SELECT poll_key,
        subject,
        content
    FROM PushRequest'
);
```

It is recommended that you include a WHERE clause in your script to filter out requests that have already been sent. For example, you can add a push request column to track the moment you inserted a request, and then use a WHERE clause in this event to filter out requests that were inserted before the last time the user synchronized.

request_delete event

This polling event accepts SQL script and is fired to perform cleanup operations when the need for push request deletion is detected. It accepts the request ID as a parameter and is executed per request ID. Your request_cursor event must contain a request ID column to use the request_delete event. You can reference the request ID using a named parameter or a question mark (?). This event is optional if you have already assigned cleanup operations to another process or event, such as the end_poll event.

The Notifier can use the DELETE statement to remove the following forms of push requests:

- **Implicitly dropped** These push requests appeared previously but did not appear in the current set of requests obtained from the request_cursor event.
- **Confirmed** These are push requests confirmed as delivered.
- **Expired** These push requests 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.

You can use the request_delete event to prevent expired or implicitly dropped requests from being deleted. For example, the CarDealer sample in the *samples-dir\MobiLink\SIS_CarDealer* directory uses the request_delete event to set the status field of the PushRequest table to 'processed'.

```
UPDATE PushRequest SET status='processed' WHERE req_id = ?
```

The begin_poll event in the sample uses the last synchronization time to check if remote devices are up-to-date before eliminating processed push requests.

See also

- “Notifier events” on page 29
- “MobiLink server settings for server-initiated synchronization” on page 25

shutdown_query event

This polling event accepts SQL script and is fired after a begin_poll event. The return value specifies the shutdown state of the Notifier. The value is null by default, so this event is not fired.

To shut down the Notifier, set up your SQL script to return 'yes'; otherwise, set it to return 'no'. If the Notifier shuts down, the end_poll event is not fired.

When storing the shutdown state in a table, use the end_connection event to reset the state.

See also

- “end_connection event” on page 36
- “Notifier events” on page 29
- “MobiLink server settings for server-initiated synchronization” on page 25

Example

The following example uses the ml_add_property system procedure to create a shutdown_query event script for a custom Notifier named Simple. The SELECT statement informs the Notifier to shut down if the tooManyNotifierErrors method returns true.

```
CALL ml_add_property('SIS', 'Notifier(Simple)', 'shutdown_query',
    'SELECT
        IF tooManyNotifierErrors() THEN
            'yes'
        ELSE
            'no'
        ENDIF'
);
```

Connection events

Connection events are a classification of Notifier events that are fired during the Notifier database connection.

begin_connection event

This event accepts SQL script and is fired after the Notifier connects to the consolidated database, but before it checks for push requests. The value is null by default, so this event is not fired.

You can use this event to create temporary tables or variables. You should not use this event to change isolation levels. To control isolation levels, use the isolation property. See "Notifier properties" on page 41.

If the Notifier loses the connection to the consolidated database, it re-runs this event immediately after reconnecting.

See also

- "Notifier events" on page 29
- "MobiLink server settings for server-initiated synchronization" on page 25

end_connection event

This event accepts SQL script and is fired just before the Notifier disconnects from the consolidated database. The value is null by default, so this event is not fired.

You can use this event to clean up temporary storage, such as SQL variables and temporary tables.

See also

- "Notifier events" on page 29
- "MobiLink server settings for server-initiated synchronization" on page 25

Asynchronous events

Asynchronous events are a classification of Notifier events that can be fired at any time during the synchronization process.

confirmation_handler event

Configure this event to handle delivery confirmation information uploaded by MobiLink Listeners. If the status parameter returns 0, then the push request identified by request_id was successfully received by the MobiLink Listener identified by the remote_device parameters.

You can use the request_option parameter to initiate an action in response to the delivery confirmation. If request_option is 0, the confirmation_handler event initiates the default action, where the request_delete event is executed to delete the original push request. If the device sending the delivery confirmation does not match the device identified by the request_id, the default action is to send the original push request through a secondary gateway.

Note
Use the dblsn -x option to allow MobiLink Listeners to upload delivery confirmation information. Use the dblsn -ni option if you want delivery confirmation but do not want IP tracking. See "MobiLink Listener options for Windows devices" on page 50.

Note
This event requires the use of a system procedure. You can not configure this event directly using the Sybase Central method. See "MobiLink server settings for server-initiated synchronization" on page 25.

The following parameters can be captured using the confirmation_handler event:

Script parameter	Type	Description
request_option (out)	Integer	Controls what the Notifier does to the request after the handler returns. The following values can be returned: • **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. • **1**: Do nothing. • **2**: Execute Notifier.request_delete. • **3**: Attempt to deliver to a secondary gateway.

Script parameter	Type	Description
status (in)	Integer	The situation summary. The status can be used during development to identify problems such as incorrect filters and handler attributes. The following values can be returned: ● **0**: Received and confirmed. ● **-2**: Right respondent but the message was rejected. ● **-3**: Right respondent and the message was accepted but the action failed. ● **-4**: Wrong respondent and the message was accepted. ● **-5**: Wrong respondent and the message was rejected. ● **-6**: Wrong respondent. The message was accepted and the action succeeded. ● **-7**: Wrong respondent. The message was accepted but the action failed.
request_id (in)	Integer	The request ID. Your request_cursor event must contain a request ID column to use the confirmation_handler event.
remote_code (in)	Integer	The summary reported by the MobiLink Listener. The following values can be returned: ● **1**: Message accepted. ● **2**: Message rejected. ● **3**: Message accepted and action succeeded. ● **4**: Message accepted and action failed.
remote_device (in)	Varchar	The device name of the responding MobiLink Listener.
remote_mluser (in)	Varchar	The MobiLink user name of the responding MobiLink Listener.
remote_action_return (in)	Varchar	The return code of the remote action.
remote_action (in)	Varchar	Reserved for the action command.
gateway (in)	Varchar	The gateway associated with the request.
address (in)	Varchar	The address associated with the request.
subject (in)	Varchar	The subject associated with the request.

Script parameter	Type	Description
content (in)	Varchar	The content associated with the request.

See also

- "Notifier events" on page 29
- "MobiLink server settings for server-initiated synchronization" on page 25
- "Gateway properties" on page 42

Example

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

```
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,
            @remote_code,
            @remote_device,
            @remote_mluser,
            @remote_action_return,
            @remote_action,
            @gateway,
            @address,
            @subject,
            @content
        );
        SET @request_option = 0;
    END
```

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

```
call ml_add_property(
    'SIS',
    'Notifier(myNotifier)',
    'confirmation_handler',
    'call CustomConfirmation(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)');
```

Alternatively, you can call this event by adding the following line to a Notifier configuration file:

```
Notifier(myNotifier).confirmation_handler = call
CustomConfirmation(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

Run the file using the mlsrv12 -notifier option. For more information on how to configure a Notifier configuration file, see "Configuring server-side settings using the Notifier configuration file" on page 28.

Common properties

Common properties are shared between Notifiers, gateways, and carriers. All common properties are optional. For more information on setting these properties, see "MobiLink server settings for server-initiated synchronization" on page 25.

Property	Value	Description
verbosity	{ **0** | **1** | **2** | **3** }	Specifies the verbosity level for Notifiers, gateways, and carriers. The following values can be used: • **0**: No trace. • **1**: Startup, shutdown, and property trace. • **2**: Display notifications. • **3**: Full-level trace. The default value is **0**.

Notifier properties

Notifier properties allow you to change the behavior of a Notifier. All Notifier properties are optional. For more information on setting these properties, see "MobiLink server settings for server-initiated synchronization" on page 25.

Property	Value	Description
connect_string	*connection_string*	Overrides the default connection behavior used to connect to a database. The default value is **ianywhere.ml.script.ServerContext**, which uses the connection string specified in the mlsrv12 command line. 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.
enable	{ **yes** | **no** }	Specifies whether the Notifier should be enabled. All enabled Notifiers start when you run the -notifier mlsrv12 option.
gui	{ **yes** | **no** }	Specifies whether the Notifier window should be displayed while the Notifier is running. The default value is **yes**. This Notifier window 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.
isolation	{ **0** | **1** | **2** | **3** }	Specifies the isolation level of the Notifier's database connection. The following values can be used: • **0**: Read uncommitted. • **1**: Read committed. • **2**: Repeatable read. • **3**: Serializable. The default value is **1**. Higher levels increase contention, but could adversely affect performance. Isolation level **0** allows reads of uncommitted data, which could get rolled back.

Property	Value	Description
poll_every	*number*{ **s** | **m** | **h** }	Specifies the amount of time to wait before confirmations timeout. The following is a list of acceptable time units: ● **s**: Denotes seconds. ● **m**: Denotes minutes. ● **h**: Denotes hours. The default value is **1m**. Time units can be combined in the *HH***h** *MM***m** *SS***s** format. If a time unit is not specified, time is measured in seconds.
shared_database_connection	{ **yes** | **no** }	Specifies whether Notifiers should share database connections. The default value is **no**. Notifiers can only share connections when their isolation levels are the same. Specify **yes** to conserve resources without incurring performance penalties. Connection sharing is not possible in some situations, such as when applications use non-unique SQL variable names among Notifiers.

Gateway properties

By default, four pre-configured gateways are created when you start the MobiLink server. They are installed when you run the MobiLink setup scripts for your consolidated database. The default gateways are named as follows:

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

You should not remove the default gateways or change their names. It is recommended that you create additional gateways with different names.

You should not need to change the properties defined in DefaultSYNC and DefaultUDP, but you must provide SMTP server information to the DefaultSYNC gateway. You should use the default gateways but, if required, you can use an alternative configuration. This section provides procedures for customizing gateway properties.

Device tracking gateway properties

Device tracking gateway properties allow you to change the behavior of a device tracking gateway. All device tracking gateway properties are optional. For more information on setting these properties, see "MobiLink server settings for server-initiated synchronization" on page 25.

Property	Value	Description
confirm_action	{ **yes** | **no** }	Specifies whether confirmation is sent on delivery through this gateway. The default value is **no**.
confirm_delivery	{ **yes** | **no** }	Specifies whether the MobiLink Listener should confirm with the consolidated database that the message was received. The default value is **yes**. The MobiLink Listener must be started with the -x MobiLink Listener option specified.
description	*description_text*	Describes the gateway.
enable	{ **yes** | **no** }	Specifies whether the device tracking gateway should be used.
smtp_gateway	*smtp_gateway_name*	Specifies the name of the SMTP subordinate gateway. The default value is **DefaultSMTP**. A device tracking gateway can only use one SMTP gateway. The gateway must be enabled.
sync_gateway	*sync_gateway_name*	Specifies the name of the SYNC subordinate gateway. The default value is **DefaultSYNC**. A device tracking gateway can only use one SYNC gateway. The gateway must be enabled.
udp_gateway	*udp_gateway_name*	Specifies the name of the UDP subordinate gateway. The default value is **DefaultUDP**. A device tracking gateway can only use one UDP gateway. The gateway must be enabled.

SMTP gateway properties

SMTP gateway properties allow you to change the behavior of an SMTP gateway. The server property is required; all other SMTP gateway properties are optional. For more information on setting these properties, see "MobiLink server settings for server-initiated synchronization" on page 25.

Property	Value	Description
confirm_action	{ **yes** | **no** }	Specifies whether confirmation is sent on delivery through this gateway. The default value is **no**.
confirm_delivery	{ **yes** | **no** }	Specifies whether this gateway confirms delivery. The default value is **no**.
confirm_timeout	*number*{ **s** | **m** | **h** }	Specifies the amount of time to wait before confirmations timeout. The following is a list of acceptable time units: • **s**: Denotes seconds. • **m**: Denotes minutes. • **h**: Denotes hours. The default value is **1m**. Time units can be combined in the *HH***h** *MM***m** *SS***s** format. If a time unit is not specified, time is measured in seconds.
description	*description_text*	Describes the gateway.
enable	{ **yes** | **no** }	Specifies whether the SYNC gateway should be used.
Listeners_are_900	{ **yes** | **no** }	Specifies whether all MobiLink Listeners are Adaptive Server Anywhere 9.0.0 clients. The default value is **no**. For Adaptive Server Anywhere 9.0.1 clients or later, leave this value at **no**.
password	*password*	Specifies the password for the SMTP service. This is required by some services.
sender	*SMTP_address*	Specifies the sender address of SMTP push notifications. The default value is **anonymous**.
server	*IP_address_or_hostname*	Specifies the IP address or host name of the SMTP server used to send messages to a MobiLink Listener. The default value is **mail**.
user	*username*	Specifies the user name of the SMTP service. This is required by some services.

SYNC gateway properties

SYNC gateway properties allow you to change the behavior of a SYNC gateway. All SYNC gateway properties are optional. For more information on setting these properties, see "MobiLink server settings for server-initiated synchronization" on page 25.

Property	Value	Description
confirm_action	{ **yes** | **no** }	Specifies whether confirmation is sent on delivery through this gateway. The default value is **no**.
confirm_delivery	{ **yes** | **no** }	Specifies whether this gateway confirms delivery. The default value is **no**.
confirm_timeout	*number*{ **s** | **m** | **h** }	Specifies the amount of time to wait before confirmations timeout. The following is a list of acceptable time units: • **s**: Denotes seconds. • **m**: Denotes minutes. • **h**: Denotes hours. The default value is **1m**. Time units can be combined in the *HH***h** *MM***m** *SS***s** format. If a time unit is not specified, time is measured in seconds.
description	*description_text*	Describes the gateway.
enable	{ **yes** | **no** }	Specifies whether the SYNC gateway should be used.
Listeners_are_900	{ **yes** | **no** }	Specifies whether all MobiLink Listeners are Adaptive Server Anywhere 9.0.0 clients. The default value is **no**. Leave this value at **no** for Adaptive Server Anywhere 9.0.1 clients or later.

UDP gateway properties

UDP gateway properties allow you to change the behavior of an UPD gateway, such as the IP address and the port number. All UDP gateway properties are optional. For more information on setting these properties, see "MobiLink server settings for server-initiated synchronization" on page 25.

Property	Value	Description
confirm_action	{ **yes** | **no** }	Specifies whether confirmation is sent on delivery through this gateway. The default value is **no**.
confirm_delivery	{ **yes** | **no** }	Specifies whether this gateway confirms delivery. The default value is **yes**.

Property	Value	Description
confirm_timeout	*number*{ **s** | **m** | **h** }	Specifies the amount of time to wait before confirmations timeout. The following is a list of acceptable time units: • **s**: Denotes seconds. • **m**: Denotes minutes. • **h**: Denotes hours. The default value is **1m**. Time units can be combined in the *HH***h** *MM***m** *SS***s** format. If a time unit is not specified, time is measured in seconds.
description	*description_text*	Describes the gateway.
enable	{ **yes** | **no** }	Specifies whether the UDP gateway should be used.
Listeners_are_900	{ **yes** | **no** }	Specifies whether all MobiLink Listeners are Adaptive Server Anywhere 9.0.0 clients. The default value is **no**. For Adaptive Server Anywhere 9.0.1 clients or later, leave this value at **no**.
Listener_port	*port_number*	Specifies the port that the remote devices uses to send UDP packets. The default value is **5001**.
sender	*IP_address_or_hostname*	Used for multi-homed hosts only. Specifies the IP address or hostname of the sender. The default value is **localhost**.
sender_port	*port_number*	Specifies the port number used to send UDP packets. By default, a free port number is randomly assigned by the operating system.

Carrier properties

Carrier properties allow you to change the behavior of a wireless carrier configuration, which provides information on mapping automatically-tracked phone numbers and network providers to email addresses. All carrier properties are optional, and are only required if you are using an SMTP gateway.

For more information on setting these properties, see "MobiLink server settings for server-initiated synchronization" on page 25.

Property	Value	Description
enable	{ **yes** | **no** }	Specifies whether the carrier should be used.

Property	Value	Description
description	*description_text*	Describes the carrier.
network_provider_id	*id_text*	Specifies the network provider ID. To use SMS on Windows Mobile Phone edition, set this property to **_generic_**.
sms_email_domain	*domain_name*	Specifies the domain name of the carrier.
sms_email_user_prefix	*prefix_name*	Specifies the prefix used in email addresses.

MobiLink Listener utility for Windows devices (dblsn)

This section provides information about MobiLink Listener utility for Windows devices, including information about Listening libraries, Listener options and keywords, Listener action commands and variables, and Listener configuration.

Syntax

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

message-handler :
[*polling-option*;...] [*filter*;...]*action*; [*option*;...]

polling-option :
[**;poll_connect =** *string*]
[**;poll_notifier =** *string*]
[**;poll_key =** *string*]
[**;poll_every =** *number*]

option :
[**;continue = yes**]
[**;confirm_action = yes**]
[**;confirm_delivery = no**]
[**;maydial = 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*

See also

- "MobiLink Listener options for Windows devices" on page 50
- "MobiLink Listener keywords for Windows devices" on page 62
- "MobiLink Listener action commands for Windows devices" on page 65
- "MobiLink Listener action variables for Windows devices" on page 68

Listening libraries for Windows devices

The MobiLink Listener comes with a UDP listening library, *lsn_udp12.dll*, which is loaded by default.

When using an SMTP gateway, you must specify an SMTP listening library. You can specify a library with the dblsn -d option, and specify library options with the dblsn -a option.

UDP (lsn_udp12.dll)

The following is a list of options supported by the UDP listening library:

Option	Description
Port= *port-number*	This option specifies the port number to listen to. The default port is **5001**.
Timeout= *seconds*	This option specifies the maximum blocking time of a read operation on the UDP listening port. This value must be smaller than the polling interval of the UDP listening thread. The default is **0**.
ShowSenderPort	This option reveals the sender port number in all occurrences of the $sender action variable. By default, the port number is hidden. When this option is specified, the port number is appended at the end of the sender address with the **:***port-number* syntax.
HideWSAErrorBox	Suppresses the error window showing errors on socket operations.
CodePage= *number*	Multibyte characters are translated into Unicode based on this number. This option only applies to Windows Mobile devices.

See also

- "-d dblsn option" on page 54
- "-a dblsn option" on page 53

MobiLink Listener options for Windows devices

The following options can be used to configure the MobiLink Listener:

Option	Description
@{ *variable* | *filename* }	Applies MobiLink Listener options from the specified environment variable or text file. See "@ dblsn option" on page 52.
-a *value*	Specifies a single library option for a listening library. See "-a dblsn option" on page 53.
-d *filename*	Specifies a listening library. See "-d dblsn option" on page 54.
-e *device-name*	Specifies the device name. See "-e dblsn option" on page 54.
-f *string*	Specifies extra information about the device. See "-f dblsn option" on page 54.
-gi *seconds*	Specifies the IP tracker polling interval. See "-gi dblsn option" on page 55.
-i *seconds*	Specifies the polling interval for SMTP connections. See "-i dblsn option" on page 55.
-l " *keyword=value;...* "	Defines and creates a message handler. See "-l dblsn option" on page 55.
-m	Turns on message logging. See "-m dblsn option" on page 56.
-ni	Disables IP tracking. See "-ni dblsn option" on page 56.
-ns	Disables SMS listening. See "-ns dblsn option" on page 56.
-nu	Disables UDP listening. See "-nu dblsn option" on page 57.
-o *filename*	Logs output to a file. See "-o dblsn option" on page 57.
-os *bytes*	Specifies the maximum size of the log file. See "-os dblsn option" on page 57.
-ot *filename*	Truncates a file, then logs output to that file. See "-ot dblsn option" on page 57.

Option	Description
-p	Allows the device to shut down automatically when idle. See "-p dblsn option" on page 58.
-pc { **+** | **-** }	Enables or disables persistent connections. See "-pc dblsn option" on page 58.
-q	Runs the MobiLink Listener in quiet mode. See "-q dblsn option" on page 59.
-r *filename*	Identifies a remote database involved in the responding action of a message filter. See "-r dblsn option" on page 59.
-sv *script-version*	Specifies the script version used for authentication. See "-sv dblsn option" on page 59.
-t {**+** | **-** } *name*	Registers or unregisters the remote ID for a remote database. See "-t dblsn option" on page 60.
-u *username*	Specifies a MobiLink user name. See "-u dblsn option" on page 60.
-v { **0** | **1** | **2** | **3** }	Specifies the verbosity level for the message log. See "-v dblsn option" on page 61.
-w *password*	Specifies a MobiLink password. See "-w dblsn option" on page 61.
-x { **http** | **https** | **tcpip** } [**(** *protocol-option=value*;... **)**]	Specifies the network protocol, and the MobiLink server protocol options. See "-x dblsn option" on page 62.
-y *newpassword*	Specifies a new MobiLink password. See "-y dblsn option" on page 62.

@ dblsn option

Applies MobiLink Listener options from the specified environment variable or text file.

Syntax

dblsn @{ *variable* | *filename* } ...

Remarks

By default, *dblsn.txt* is the argument file when the MobiLink Listener is run without parameters.

If a file and an environment variable with the same name exist, the environment variable is used.

Use the File Hiding utility to obfuscate passwords and other sensitive information in the text file. See "File Hiding utility (dbfhide)" [*SQL Anywhere Server - Database Administration*].

See also

- "Using configuration files" [*SQL Anywhere Server - Database Administration*]

Example

A sample text file is located at ***samples-dir**MobiLink**SIS_SimpleListener**dblsn.txt***.

The following example stores command line options in a **dblsnoptions** environment variable:

```
dblsn @dblsnoptions
```

The following example stores command line options in **mydblsn.txt**, a fully qualified text file:

```
dblsn @mydblsn.txt
```

-a dblsn option

Specifies a single library option for a listening library.

Syntax

dblsn -a *value* ...

Remarks

By default, the MobiLink Listener uses *lsn_udp12.dll* if a library is not specified. Use the -d option to specify alternative or additional libraries.

Use the **?** value to view all available library options.

Use the -a option multiple times to set additional library options.

See also

- "Listening libraries for Windows devices" on page 50
- "-d dblsn option" on page 54

Example

The following example specifies the port option and declares the ShowSenderPort option in the *lsn_udp12.dll* listening library:

```
dblsn -d lsn_udp12.dll -a port=1234 -a ShowSenderPort
```

The following example specifies the port option for two different libraries:

```
dblsn -d lsn_udp12.dll -a port=1234 -d maac750.dll -a port=2345
```

The following example displays all available library options in the default library:

```
dblsn -a ?
```

-d dblsn option

Specifies a listening library.

Syntax

dblsn -d *filename* ...

Remarks

By default, the MobiLink Listener uses the *lsn_udp12.dll* listening library.

Use the -d option multiple times to enable multi-channel listening, which enables listening on multiple media.

See also

- "Listening libraries for Windows devices" on page 50

Example

The following example specifies the *maac750.dll* listening library:

```
dblsn -d maac750.dll
```

-e dblsn option

Specifies the device name.

Syntax

dblsn -e *device-name* ...

Remarks

By default, the device name is automatically generated by the operating system.

When connecting to the MobiLink server, make sure that all device names are unique.

The device name can be found in the MobiLink Listener window.

-f dblsn option

Specifies extra information about the device.

Syntax

dblsn -f *string* ...

Remarks

By default, this information is the version number of the operating system running on the device.

-gi dblsn option

Specifies the IP tracker polling interval.

Syntax

dblsn -gi *number* ...

Remarks

By default, the IP tracker polls every 60 seconds.

-i dblsn option

Specifies the polling interval for SMTP connections.

Syntax

dblsn -i *number* ...

Remarks

The -i option specifies the frequency at which the MobiLink Listener checks for messages.

For SMTP connections, the default is 30 seconds. For UDP connections, the MobiLink Listener connects immediately.

The -i option can be used once for every listening library that is specified with the -d option.

See also

- "-d dblsn option" on page 54

Example

The following example specifies the polling intervals for two different libraries:

```
dblsn -d lsn_udp12.dll -i 60 -d maac750.dll -i 45
```

-l dblsn option

Defines and creates a message handler.

Syntax

dblsn -l "*keyword=value;...***"** ...

Remarks

For a list of acceptable keywords, see "MobiLink Listener keywords for Windows devices" on page 62.

Use the -l option multiple times to define additional message handlers for push notifications. Message handlers are processed in the order they are specified.

See also

- "Message handlers" on page 12

-m dblsn option

Turns on message logging.

Syntax

dblsn -m ...

Remarks

By default, message logging is turned off.

-ni dblsn option

Disables IP tracking.

Syntax

dblsn -ni ...

Remarks

By default, IP tracking is enabled.

This option does not stop delivery confirmation.

The -ni option disables UDP address tracking when used in conjunction with the -x option. This feature is useful if you want device tracking to exclude UDP address updates.

See also

- "-x dblsn option" on page 62

-ns dblsn option

Disables SMS listening.

Syntax

dblsn -ns ...

Remarks

By default, SMS listening is enabled for Windows Mobile 2003 and later.

-nu dblsn option

Disables UDP listening.

Syntax

dblsn -nu ...

Remarks

By default, UDP listening is enabled.

-o dblsn option

Logs output to a file.

Syntax

dblsn -o *filename* ...

Remarks

By default, output is logged to the MobiLink Listener window.

See also

- "-ot dblsn option" on page 57

-os dblsn option

Specifies the maximum size of the log file.

Syntax

dblsn -os *bytes* ...

Remarks

By default, there is no maximum size limit. The minimum size limit is 10000.

-ot dblsn option

Truncates a file, then logs output to that file.

Syntax

dblsn -ot *filename* ...

Remarks

The file contents are deleted before output is logged.

See also

- "-o dblsn option" on page 57

-p dblsn option

Allows the device to shut down automatically when idle.

Syntax

dblsn -p ...

Remarks

By default, the MobiLink Listener prevents the device from shutting down.

This option applies to Windows Mobile devices only.

-pc dblsn option

Syntax

Enables or disables persistent connections.

dblsn -pc { **+** | **-** } ...

Remarks

By default, persistent connections are enabled. The **-** flag disables persistent connections; the **+** flag enables them.

Disabling persistent connections prevents the MobiLink Listener from receiving push notifications, but allows short-lived persistent connections for device tracking and confirmation.

If a persistent connection is broken, the MobiLink Listener continuously attempts to reconnect.

Example

The following example disables persistent connections:

```
dblsn -pc-
```


-q dblsn option

Runs the MobiLink Listener in quiet mode.

Syntax

dblsn -q ...

Remarks

The -q option minimizes the MobiLink Listener window. By default, the MobiLink Listener window is displayed.

-r dblsn option

Identifies a remote database involved in the responding action of a message filter.

Syntax

dblsn -r *filename* ...

Remarks

The *filename* must contain the full path of an RID file. This file is automatically created by dbmlsync after the first synchronization. It uses the same location and name as the database file. For UltraLite databases, *filename* should be the same as the database name.

When applying the -r option, the $remote_id action variable can be used in message handlers to reference the remote ID in the RID file. By default, Remote IDs are GUIDs.

Use the -r option multiple times to identify multiple databases.

See also

- "Working with message handlers" on page 13
- "MobiLink Listener action commands for Windows devices" on page 65

-sv dblsn option

Specifies the script version used for authentication.

Syntax

dblsn -sv *script-version* ...

Remarks

By default, the MobiLink Listener uses the ml_global server script version if it is defined.

-t dblsn option

Registers or unregisters the remote ID for a remote database.

Syntax

dblsn -t { **+** | **-** } *name* ...

Remarks

The + flag registers a remote ID; the - flag unregisters ID.

Registering allows the MobiLink Listener to address push notifications by referencing that remote ID.

When device tracking information is uploaded successfully, registered IDs are retained in the ml_listening system table on the server. You only need to register the ID once.

Use the -t option multiple times to register or unregister multiple IDs. Registering multiple IDs is useful for addressing push notifications to multiple remote databases.

See also

- "MobiLink Listener action commands for Windows devices" on page 65

-u dblsn option

Specifies a MobiLink user name for the MobiLink Listener.

Syntax

dblsn -u *username* ...

Remarks

By default, the user name is *device-name*-**dblsn**, where *device-name* is the name of your device. You can specify a device name using the -e option.

The MobiLink Listener uses the user name to connect to the MobiLink server for device tracking, confirmations, and persistent connections.

The user name must be a unique MobiLink user name registered with the MobiLink server. The name must exist in the ml_user system table on the consolidated database.

See also

- "-e dblsn option" on page 54
- "-w dblsn option" on page 61
- "Creating and registering MobiLink users" [*MobiLink - Client Administration*]

-v dblsn option

Specifies the verbosity level for the message log.

Syntax

dblsn -v { **0** | **1** | **2** | **3** } ...

Remarks

By default, the verbosity level is set to 0.

The following table summarizes the possible verbosity level values.

Verbosity level	Description
0	Verbosity is turned off.
1	Displays listening library messages, basic action tracing steps, and command line options.
2	Displays level 1 verbosity messages, and detailed action tracing steps.
3	Displays level 2 verbosity messages, polling states, and listening states.

You must use the -m option to output push notifications to the message log.

See also

- "-m dblsn option" on page 56
- "Log database server messages to a file" [*SQL Anywhere Server - Database Administration*]

-w dblsn option

Specifies a MobiLink password.

Syntax

dblsn -w *password* ...

Remarks

Passwords must be registered with the MobiLink server under the associated MobiLink user name.

The MobiLink Listener uses the password to connect to the MobiLink server for device tracking, confirmations, and persistent connections.

See also

- "-u dblsn option" on page 60

-x dblsn option

Specifies the network protocol, and the MobiLink server protocol options.

Syntax

dblsn -x { **http** | **https** | **tcpip** } [**(***protocol-option***=***value*;...**)**] ...

Remarks

A connection to the MobiLink server is required so the MobiLink Listener can send device tracking information and delivery confirmation to the consolidated database. Use the **host** protocol option to specify the location of the MobiLink server. See "host" [*MobiLink - Client Administration*].

The -x option allows the device to update the consolidated database if the server address changes.

See also

- "MobiLink client network protocol options" [*MobiLink - Client Administration*]

-y dblsn option

Specifies a new MobiLink password.

Syntax

dblsn -y *newpassword* ...

Remarks

The -y option is not applicable if your authentication system does not allow remote devices to change their passwords.

MobiLink Listener keywords for Windows devices

The following keywords can be used to configure message handlers created with the dblsn -l option. For more information about using MobiLink Listener keywords, see "-l dblsn option" on page 55.

Filter keywords

Use the following keywords to filter messages in a push notification:

Keyword syntax	Description
subject= *subject-string*	Filters a message if the subject is textually equivalent to *subject-string*.

Keyword syntax	Description
content= *content-string*	Filters a message if the content is textually equivalent to *content-string*.
message= *message-string*	Filters a message if the entire message is textually equivalent to *message-string*.
message_start= *message-string*	Filters a message if it begins with *message-string*.
sender= *sender-string*	Filters a message if it is send by *sender-string*.

Action keywords

Use the following keywords to initiate an action when a filter condition is met:

Keyword syntax	Description
action= *command*	Specifies an action command. See "MobiLink Listener action commands for Windows devices" on page 65.
altaction= *command*	Specifies an alternative action command that is initiated when the action command fails. See "MobiLink Listener action commands for Windows devices" on page 65.

Polling options

Use the following options to configure a light weight poller:

Keyword syntax	Description
poll_connect={ **http** | **https** | **tcpip** } [**(***protocol-option=value*;...**)**]	Specifies the light weight network protocol options required to connect to the server. The default value is inherited from the dblsn -x option. See "-x dblsn option" on page 62.
poll_notifier= *Notifier-string*	Specifies the name of the Notifier that handles push requests. Required.
poll_key= *key-string*	Specifies the name of the MobiLink Listener to identify itself to the Notifier. This value must be unique. Required.

Keyword syntax	Description
poll_every= *seconds-number*	Specifies how often the MobiLink Listener should poll the server. The interval is measured in seconds. The default value is auto-retrieved from the MobiLink server.

Options

The following can be used to configure message handler behavior:

Keyword syntax	Description
continue=[**yes** | **no**]	Specifies whether the MobiLink Listener should continue listening after finding the first match. The default value is **no**. A **yes** value is useful when specifying multiple filters, where one message initiates multiple actions.
confirm_action=[**yes** | **no**]	Specifies whether the filter should confirm the action. The default value is **yes**.
confirm_delivery=[**yes** | **no**]	Specifies whether the filter should confirm qualified message delivery. The default value is **yes**, so delivery confirmation is sent after the first filter accepts the message. Delivery can only be confirmed if the message requires confirmation, and if the filter accepts the message. A message requires confirmation if the specified gateway has its confirm_delivery keyword value set to **yes**. A **no** value can be used when multiple filters accept the same message to give you finer control over which filter should confirm the delivery. For information about handling delivery confirmation on the server, see “confirmation_handler event” on page 37.
maydial=[**yes** | **no**]	Specifies whether the action has permission to dial the modem. The default value is **yes**. A **no** value causes the MobiLink Listener to release the modem before the action.

See also

- “Message handlers” on page 12
- “MobiLink Listener action commands for Windows devices” on page 65

MobiLink Listener action commands for Windows devices

An action is specified when you configure a new message handler. When a filter condition is met, an action is initiated. If the action fails, an alternative action is initiated. Actions are defined using the **action** keyword; alternative actions are defined using the **altaction** keyword. For more information about MobiLink Listener usage, see "MobiLink Listener keywords for Windows devices" on page 62.

Following is a list of action commands:

Command	Description
START *program arglist*	Initiates a program while the MobiLink Listener runs in the background. See "START action command" on page 65.
RUN *program arglist*	Pauses the MobiLink Listener to run a program. See "RUN action command" on page 66.
POST *windowmessage* | *id* to *windowclass* | *windowtitle*	Posts a window message to a window class. See "POST action command" on page 66.
SOCKET port=*windowname*[;**host**=*hostname*][;**sendText**=*text*][;**recvText**=*text*[;**timeout**=*seconds*]]	Sends a message to an application using a TCP/IP connection. See "SOCKET action command" on page 67.
DBLSN FULL SHUTDOWN	Forces the MobiLink Listener to shutdown. See "DBLSN FULL SHUTDOWN action command" on page 68.

You can only specify one action per **action** or **altaction** keyword. If you want an action to perform multiple tasks, create a batch file that contains multiple commands and run the file using the **RUN** action command.

See also

- "Initiating actions" on page 14

START action command

Initiates a program while the MobiLink Listener runs in the background.

Syntax

action='START *program arglist*'

Remarks

When you start a program, the MobiLink Listener continues listening for more push notifications.

The MobiLink Listener does not wait for the program to finish, so it can only determine if an action command has failed, or if it cannot start the specified program.

Example

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

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

RUN action command

Pauses the MobiLink Listener to run a program.

Syntax

action='RUN *program arglist*'

Remarks

The MobiLink Listener waits for the program to finish, then resumes listening.

When running a program, the MobiLink Listener determines if the program has failed, if the MobiLink Listener cannot start the program or if the program returns a non-zero return code.

Example

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

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

POST action command

Posts a window message to a window class.

Syntax

action='POST *windowmessage* | *id* to *windowclass* | *windowtitle*'

Remarks

The POST command can be used to signal applications that use window messages.

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

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

If your window message or window class name contains non-alphanumeric characters, such as spaces or punctuation marks, encapsulate the text in single quotes ('). The escape character is also a single quote, so if your window message or window class name contains single quotes, reference the quote using two single quotes ('').

The following are valid for POST:

- **Post by decimal id** For example, `post 999 to <wc|wt>`
- **Post by hex id** For example, `post 0x3E7 to <wc|wt>`
- **Post by registered message name** For example, `post myRegisteredMsgName to <wc|wt>`

Example

To demonstrate the use of windows and messages containing single quotes, the following example posts mike's_message window message to the mike's_class window class:

```
dblsn -l "action='post mike''s_message to mike''s_class';"
```

The following example posts a window message, dbas_synchronize, to a dbmlsync instance registered with the dbmlsync_FullSync class name:

```
dblsn -l "action='post dbas_synchronize to dbmlsync_FullSync';"
```

See also

- "-wc dbmlsync option" [*MobiLink - Client Administration*]

SOCKET action command

Sends a message to an application using a TCP/IP connection.

Syntax

action='SOCKET port=*windowname*[;**host=***hostname*][;**sendText=***text*]
[;**recvText=***text*[;**timeout=***seconds*]]**'**

Remarks

The SOCKET command is used for passing dynamic information to a running application, and for integrating messages into Java and Visual Basic applications. Both languages do not support custom window messaging, and eMbedded Visual Basic does not support command line parameters.

To connect to a socket, you must specify the port and the host. Use sendText to input your message.

Use recvText to display a message when confirming that sendText is successfully received by the application. When using recvText, you can specify a timeout limit. The action fails if the MobiLink Listener can not connect, send acknowledgements, or receive acknowledgements during the timeout limit.

Example

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

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

DBLSN FULL SHUTDOWN action command

Forces the MobiLink Listener to shutdown.

Syntax

action='DBLSN FULL SHUTDOWN'

Remarks

After shutdown, the MobiLink Listener stops handling push notifications and stops synchronizing device tracking information. You must restart the MobiLink Listener to continue with server-initiated synchronization.

MobiLink Listener action variables for Windows devices

The following action variables can be used in an action or a filter. The value is substituted into the action variable before initiating the message handler.

Action variables start with a dollar sign ($). The escape character is also a dollar sign, so use two dollar signs ($$) to specify a single dollar sign as plain text.

Variable	Description
$subject	The subject of a message.
$content	The content of a message.
$message	An entire message, including subject, content, and sender.

Variable	Description
$message_start	The beginning portion of a message, as specified by the message_start filter keyword. This variable is only available if you have specified the message_start filter keyword. See "MobiLink Listener keywords for Windows devices" on page 62.
$message_end	The portion of a message, excluded from the message_start filter keyword. This variable is only available if you have specified the message_start filter keyword. See "MobiLink Listener keywords for Windows devices" on page 62.
$ml_connect	The MobiLink network protocol options as specified by the dblsn -x option. The default is an empty string. See "-x dblsn option" on page 62.
$ml_user	The MobiLink user name as specified by the dblsn -u option. The default name is *device-name*-**dblsn**.
$ml_password	The MobiLink password as specified by dblsn -w option, or the new MobiLink password if -y is used.
$priority	The meaning of this variable is carrier library-dependent.
$request_id	The request ID that was specified in a push request. See "Push requests" on page 5.
$remote_id	The remote ID. This variable can only be used when the dblsn -r option is specified. See "Filtering messages by remote ID" on page 16.
$sender	The sender of a message.
$type	The meaning of this variable is carrier library-dependent.
$year	The meaning of this variable is carrier library-dependent.
$month	The meaning of this variable is carrier library-dependent. Values can be from 1-12.
$day	The meaning of this variable is carrier library-dependent. Values can be from 1-31.
$hour	The meaning of this variable is carrier library-dependent. Values can be from 0-23.
$minute	The meaning of this variable is carrier library-dependent. Values can be from 0-59.
$second	The meaning of this variable is carrier library-dependent. Values can be from 0-59.

Variable	Description
$best_adapter_mac	The MAC address of the best NIC for reaching the MobiLink server that is specified by the dblsn -x option. If the best route does not go through a NIC, the value is an empty string.
$best_adapter_name	The adapter name of the best NIC for reaching the MobiLink server that is specified by the dblsn -x option. If the best route does not go through a NIC, the value is an empty string.
$best_ip	The IP address of the best IP interface for reaching the MobiLink server that is specified by the dblsn -x option. If that server is unreachable, the value is 0.0.0.0.
$best_network_name	The RAS or dial-up profile name of the best profile for reaching the MobiLink server that is specified by the dblsn -x option. If the best route does not go through a RAS/dial-up connection, the value is an empty string.
$adapters	A list of active network adapter names, each separated by a vertical bar (|).
$network_names	A list of connected RAS entry names, each separated by a vertical bar (|). RAS entry names are sometimes referred to as dial-up entry names or Dial-Up Network (DUN).
$poll_connect	The MobiLink network protocol options as specified by the poll_connect polling keyword. The default is an empty string. See “MobiLink Listener keywords for Windows devices” on page 62.
$poll_notifier	The name of the Notifier as specified by the poll_notifier polling keyword. See “MobiLink Listener keywords for Windows devices” on page 62.
$poll_key	The poll key as specified by the poll_key polling keyword. See “MobiLink Listener keywords for Windows devices” on page 62.
$poll_every	The polling frequency as specified by the poll_every polling keyword. See “MobiLink Listener keywords for Windows devices” on page 62.

See also

- “-l dblsn option” on page 55
- “MobiLink Listener keywords for Windows devices” on page 62
- “MobiLink Listener action commands for Windows devices” on page 65

Example

The following examples uses the $message_end action variable to determine which publication to synchronize:

```
dblsn -l "message_start=start-of-message;action='run dbmlsync.exe -c ... -n
$message_end'"
```


Light weight polling API

The light weight polling API is a programming interface that you can integrate into your device application. It contains the methods needed to poll a server.

Required files

All directories are relative to *install-dir* . The following is a list of files required to compile the light weight polling API:

File name or location	Description
Bin32\mllplib12.dll	Light weight polling API runtime dynamic library.
SDK\Lib\x86\mllplib12.lib and *SDK\Lib\x64\mllplib12.lib*	Light weight polling API runtime import library.
SDK\Include\mllplib.h	Light weight polling API header file.

A SIS_CarDealer_LP_API sample application written in C is supplied in *samples-dir\MobiLink\SIS_CarDealer_LP_API*. For more information about the *samples-dir*, see "Samples directory" [*SQL Anywhere Server - Database Administration*].

API members

Method	Description
"MLLightPoller class" on page 71	Represents a light weight poller object.
"MLLPCreatePoller method" on page 75	Creates an instance of an MLLightPoller.
"MLLPDestroyPoller method" on page 75	Destroys an instance of an MLLightPoller.

MLLightPoller class

Represents a light weight poller object.

Syntax

```
public class MLLightPoller
```

Members

Name	Description
"Poll method" on page 72	Polls the server, prompting a Notifier to check a cache for push requests.
"SetConnectInfo method" on page 73	Sets up a MobiLink client stream type, and network protocol options.
"auth_status enumeration" on page 73	Specifies possible authentication status codes.
"return_code enumeration" on page 74	Specifies possible return codes.

Example

```
MLLightPoller * poller = MLLCreatePoller();
```

Poll method

Polls the server, prompting a Notifier to check a cache for push requests.

Syntax

```
public virtual return_code MLLightPoller::Poll(
    const char * notifier,
    const char * key,
    char * subject = 0,
    size_t * subjectSize = 0,
    char * content = 0,
    size_t * contentSize = 0
)
```

Parameters

- **notifier** The name of the Notifier.
- **key** The name of the poll key identifying the MobiLink Listener.
- **subject** The buffer to receive a message subject. (null-terminated)
- **subjectSize** IN: The size of the subject buffer.

  OUT: The size of the received subject, including a null-terminator, where zero indicates a null subject.
- **content** The buffer to receive message content. (null-terminated)
- **contentSize** IN: The size of the content buffer.

OUT: The size of the received content, including a null-terminator, where zero indicates null content.

Returns

One of the codes listed in the return_code enumeration. See "return_code enumeration" on page 74.

Remarks

The MobiLink Listener connects to the Notifier, then disconnects after the Notifier checks its cache for a push notification targeted for the given poll key.

SetConnectInfo method

Sets up a MobiLink client stream type, and network protocol options.

Syntax

```
public virtual return_code MLLightPoller::SetConnectInfo(
    const char * streamName,
    const char * streamParams
);
```

Parameters

- **streamName** The network protocol to use. Acceptable values are: **tcpip**, **http**, **https**, or **tls**.
- **streamParams** A protocol options string, formatted in a semicolon delimited list. For a complete list of options, see "MobiLink client network protocol options" [*MobiLink - Client Administration*].

Returns

One of the codes listed in the return_code enumeration. See "return_code enumeration" on page 74.

Example

```
poller_ret = poller->SetConnectInfo("http", "host=localhost;port=80;")
```

auth_status enumeration

Specifies possible authentication status codes.

Syntax

```
public typedef enum MLLightPoller::auth_status;
```

Members

Member name	Description
AUTH_EXPIRED	Authentication has expired.

Member name	Description
AUTH_IN_USE	The user name is already authenticated.
AUTH_INVALID	Authentication is invalid.
AUTH_UNKNOWN	Authentication is unknown.
AUTH_VALID	Authentication is valid.
AUTH_VALID_BUT_EX-PIRES_SOON	Authentication is valid but expires soon.

return_code enumeration

Specifies possible return codes.

Syntax

```
public typedef enum MLLightPoller::return_code;
```

Members

Name	Description
AUTH_FAILED	Failed to authenticate the connection to the MobiLink server.
BAD_STREAM_NAME	Unrecognizable stream name.
BAD_STREAM_PAR-AM	Failed parsing a stream parameter.
COMMUNICA-TION_ERROR	Failed due to a communication error.
CONNECT_FAILED	Failed to connect to the MobiLink server.
CONTENT_OVER-FLOW	The content buffer is too small.
KEY_NOT_FOUND	There is no push notification for the specified key from the specified Notifier.
NYI	For internal use only.
OK	Method ran successfully.
SUBJECT_OVERFLOW	The subject buffer is too small.

MLLPCreatePoller method

Creates an instance of an MLLightPoller.

Syntax

```
extern MLLightPoller * MLLPCreatePoller()
```

Returns

A new MLLightPoller object.

See also

- "MLLPDestroyPoller method" on page 75

Example

```
poller = MLLPCreatePoller();
```

MLLPDestroyPoller method

Destroys an instance of an MLLightPoller.

Syntax

```
extern void MLLPDestroyPoller(
    MLLightPoller * poller
)
```

Parameters

- **poller** The MLLightPoller to destroy.

Remarks

This method accepts null MLLightPoller objects.

See also

- "MLLPCreatePoller method" on page 75

Example

```
MLLPDestroyPoller(poller);
```


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.

See also

- “Device tracking gateways” on page 20
- “Adding support for device tracking” on page 20
- “MobiLink server system tables” [*MobiLink - Server Administration*]
- “MobiLink server system procedures” [*MobiLink - Server Administration*]

ml_delete_device system procedure

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

Parameters

Item	Parameter	Description
1	device	VARCHAR(255). Device name.

Remarks

This function is useful only if you are manually setting up device tracking. See “Adding support for device tracking” on page 20.

Example

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

```
CALL ml_delete_device('myOldDevice');
```

ml_delete_device_address system procedure

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

Parameters

Item	Parameter	Description
1	device	VARCHAR(255)
2	medium	VARCHAR(255)

Remarks

This system procedure is useful only if you are manually setting up device tracking. See "Adding support for device tracking" on page 20.

Example

Delete an address record:

```
CALL ml_delete_device_address('myWindowsMobile', 'ROGERS AT&T');
```

ml_delete_listening system procedure

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

Parameters

Item	Parameter	Description
1	ml_user	VARCHAR(128)

Remarks

This system procedure is useful only if you are manually setting up device tracking. See "Adding support for device tracking" on page 20.

Example

Delete a recipient record:

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

ml_set_device system procedure

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

Item	Parameter	Description
1	device	VARCHAR(255). User-defined unique device name.
2	listener_version	VARCHAR(128). Optional remarks on MobiLink Listener version.
3	listener_protocol	INTEGER. Use **0** for version 9.0.0 or **2** for post-9.0.0 MobiLink Listeners for Windows devices.
4	info	VARCHAR(255). Optional device information.
5	ignore_tracking	CHAR(1). Set to **y** to ignore tracking and stop it from overwriting manually entered data.
6	source	VARCHAR(255). Optional remarks on the source of this record.

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.

This system procedure is useful only if you are manually setting up device tracking. See “Adding support for device tracking” on page 20.

See also

- “ml_set_device_address system procedure” on page 79
- “ml_set_listening system procedure” on page 80

Example

For each device, add a device record:

```
CALL ml_set_device(
    'myWindowsMobile',
    'MobiLink Listeners for myWindowsMobile - 9.0.1',
    '1',
    'not used',
    'y',
    'manually entered by administrator'
);
```

ml_set_device_address system procedure

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

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

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.

This system procedure is useful only if you are manually setting up device tracking. See “Adding support for device tracking” on page 20.

See also

- “ml_set_device system procedure” on page 78
- “ml_set_listening system procedure” on page 80

Example

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

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

ml_set_listening system procedure

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

Item	Parameter	Description
1	ml_user	VARCHAR(128). MobiLink user name.
2	device	VARCHAR(255). Existing device name.
3	listening	CHAR(1). Set to **y** to activate this record to be used for DeviceTracker addressing.
4	ignore_tracking	CHAR(1). Set to **y** to ignore tracking and stop it from overwriting manually entered data.
5	source	VARCHAR(255). Optional remarks on the source of this record.

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.

This system procedure is useful only if you are manually setting up device tracking. See "Adding support for device tracking" on page 20.

See also

- "ml_set_device system procedure" on page 78
- "ml_set_device_address system procedure" on page 79

Example

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

```
CALL ml_set_listening(
    'myULDB',
    'myWindowsMobile',
    'y',
    'y',
    'manually entered by administrator'
);
```

ml_set_sis_sync_state system procedure

Use this system procedure to record the MobiLink synchronization state into the ml_sis_sync_state system table.

Parameters

Item	Parameter	Description
1	remote_id	VARCHAR(128).
2	subscription_id	VARCHAR(255).
3	publica- tion_name	VARCHAR(128).
4	user_name	VARCHAR(128).
5	last_upload	TIMESTAMP.
6	last_download	TIMESTAMP.

Remarks

Call the ml_set_sis_sync_state system procedure in the publication_nonblocking_download_ack event to allow users to create a request_cursor event that references the ml_sis_sync_state table.

See also

- "publication_nonblocking_download_ack connection event" [*MobiLink - Server Administration*]

Example

Specify a publication_nonblocking_download_ack event script with the following script to record the synchronization state:

```
CALL ml_set_sis_sync_state(
    {ml s.remote_id},
    NULL,
    {ml s.publication_name},
    {ml s.username},
    NULL,
    {ml s.last_publication_download}
);
```


Server-initiated synchronization advanced topics

The following sections provide information on advanced topics related to server-initiated synchronization.

Message syntax

The following message syntax applies to light weight polling (default), UDP gateways, and SYNC gateways:

message = **[subject]content**

Messages sent using the SMTP gateway have one of the following syntax structures:

- **message** = *sender*[*subject*]*content*
- **message** = *sender*(*subject*)*content*
- **message** = *sender*{*subject*}*content*
- **message** = *sender*<*subject*>*content*
- **message** = *sender*'*subject*'*content*
- **message** = *sender*"*subject*"*content*

The proper message syntax and *sender* email address syntax are dependent on your wireless carrier. To determine the message syntax, run the MobiLink Listener with message logging enabled, and with the verbosity level set to 2 using the dblsn -m and -v options. The message log contains the proper syntax when you initially run the MobiLink Listener.

When using a device tracking gateway, the message syntax depends on the subordinate gateway used to send the message. If you are using an SMTP subordinate gateway, the syntax is dependent on your public wireless carrier.

Remarks

Braces, chevrons, double quotations, parenthesis, single quotations, and square brackets are reserved for internal use, and should not be used within **subject**. For more information about message limitations and restrictions, see "Working with push requests" on page 7.

See also

- "MobiLink Listener keywords for Windows devices" on page 62
- "Gateways and carriers" on page 19
- "-m dblsn option" on page 56
- "-v dblsn option" on page 61

Using SA_SEND_UDP to send push notifications

If you are using a SQL Anywhere consolidated database, you can use the SA_SEND_UDP system procedure to send push notifications to a device through a UDP gateway. This method is an alternative to sending push notifications with Notifiers.

By appending a **1** to the end of your original message, and then using that message in the **msg** argument of a SA_SEND_UDP system procedure, you send the original message to a MobiLink Listener.

The following procedure assumes that a MobiLink Listener is already set up on a device, and is listening for push notifications. The following command is used on the device:

```
dblsn -l "message=RunBrowser;action='START iexplore.exe http://
www.ianywhere.com';"
```

For demonstration purposes, assume that the MobiLink Listener loads Internet Explorer whenever it receives a **RunBrowser** message. This procedure also assumes that a SQL Anywhere consolidated database is running on the MobiLink server.

To send a push notification using the SA_SEND_UDP system procedure

1. Run Interactive SQL and connect to your consolidated database.
2. Execute the following command:

   ```
   dbisql -c "dsn=consdb_source_name"
   ```

   Replace *consdb_source_name* with the ODBC name of your consolidated database.
3. Execute the following command to send the push notification:

   ```
   CALL SA_SEND_UDP('device_ip_address', 5001, 'RunBrowser1')
   ```

   The first argument ensures that the push notification is sent to the correct device. Replace *device_ip_address* with the IP address of the device. If you are running the MobiLink Listener on the same computer as the MobiLink server, use **localhost**.

   The second argument is the port number. By default, MobiLink Listeners use port 5001 to listen for UDP.

   The third argument is the message to send with a **1** appended at the end. By appending a 1, which is a reserved server-initiated synchronization protocol, the **RunBrowser** message is sent to the device using a UDP gateway.

   When the system call is executed, the **RunBrowser** message is sent to the device, causing the device to run Internet Explorer and load the iAnywhere home page.

For more information about the SA_SEND_UDP system procedure, see "sa_send_udp system procedure" [*SQL Anywhere Server - SQL Reference*].

Server-initiated synchronization tutorials

Use the following tutorials to gain a better understanding of how to use server-initiated synchronization.

Tutorial: Server-initiated synchronization using light weight polling

This tutorial demonstrates how to configure a SQL Anywhere consolidated and remote database for server-initiated synchronization. It is based on the sample code located in *samples-dir\MobiLink \SIS_CarDealer_LP_DBLSN*.

Sample implementations of server-initiated synchronization are located in *samples-dir\MobiLink*. All server-initiated synchronization sample directory names begin with the SIS_ prefix.

Required software

- SQL Anywhere 12

Competencies and experience

- Basic knowledge of MobiLink event scripts.

Goals

- Set up a SQL Anywhere consolidated database for server-initiated synchronization.
- Configure server-side properties.
- Issue push requests to prompt a server-initiated synchronization.

Suggested background reading

- "Introduction to server-initiated synchronization" on page 1

Lesson 1: Set up the consolidated database

In this lesson, you create a consolidated database named **SIS_CarDealer_LP_DBLSN_CONDB** with the scripts required for synchronization using the dbinit utility.

To create and start a new SQL Anywhere consolidated database

1. Create a new working directory to store the consolidated database.

   This tutorial assumes *c:\MLsis* as the working directory.

2. Create the SQL Anywhere consolidated database using the dbinit utility.

   Run the following command:

```
dbinit SIS_CarDealer_LP_DBLSN_CONDB
```

3. Start the consolidated database.

   Run the following command:

   ```
   dbeng12 SIS_CarDealer_LP_DBLSN_CONDB
   ```

Define an ODBC data source for the consolidated database

Use the SQL Anywhere 12 driver to define an ODBC data source for the **SIS_CarDealer_LP_DBLSN_CONDB** database.

To define an ODBC data source for the consolidated database

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **ODBC Data Source Administrator**.
2. Click the **User DSN** tab, and click **Add**.
3. In the **Create New Data Source** window, click **SQL Anywhere 12** and click **Finish**.
4. Perform the following tasks in the **ODBC Configuration For SQL Anywhere** window:
   a. Click the **ODBC** tab.
   b. In the **Data Source Name** field, type **SIS_CarDealer_LP_DBLSN_CONDB**.
   c. Click the **Login** tab.
   d. In the **User ID** field, type **DBA**.
   e. In the **Password** field, type **sql**.
   f. From the **Action** dropdown list, choose **Connect To A Running Database On This Computer**.
   g. In the **Server Name** field, type **SIS_CarDealer_LP_DBLSN_CONDB**.
   h. Click **OK**.
5. Close the ODBC data source administrator.

   Click **OK** on the **ODBC Data Source Administrator** window.

See also

- "Creating ODBC data sources" [*SQL Anywhere Server - Database Administration*]

Lesson 2: Generate database schema

In this lesson, you generate a database schema, which includes a Dealer table, a non_sync_request table, a download_cursor synchronization script. This database schema satisfies the requirements for generating push requests.

To set up the database schema

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. Perform the following tasks to connect to the consolidated database:
   a. From the **Connections** menu, choose **Connect with SQL Anywhere 12**.
   b. From the **Action** dropdown list, choose **Connect with an ODBC Data Source**.
   c. Click **ODBC Data Source Name**, and click **Browse**.
   d. Select **SIS_CarDealer_LP_DBLSN_CONDB** and then click **OK**.
   e. Click **Connect**.
3. Connect to your database using Interactive SQL.

   You can start Interactive SQL from Sybase Central or from a command prompt.

   - To start Interactive SQL from Sybase Central, right-click the **SIS_CarDealer_LP_DBLSN_CONDB - DBA** database and click **Open Interactive SQL**.
   - To start Interactive SQL at a command prompt, run the following command:

     ```
     dbisql -c "dsn=SIS_CarDealer_LP_DBLSN_CONDB"
     ```

4. Run the following SQL script to create and set up the Dealer and non_sync_request tables:

   ```
   CREATE TABLE Dealer (
       name           VARCHAR(10) NOT NULL PRIMARY KEY,
       rating         VARCHAR(5),
       last_modified  TIMESTAMP DEFAULT TIMESTAMP
   )

   CREATE TABLE non_sync_request(
       poll_key       VARCHAR(128)
   )
   ```

5. Insert data into the Dealer table using the following statements:

   ```
   INSERT INTO Dealer(name, rating) VALUES ('Audi', 'a');
   INSERT INTO Dealer(name, rating) VALUES ('Buick', 'b');
   INSERT INTO Dealer(name, rating) VALUES ('Chrysler', 'c');
   INSERT INTO Dealer(name, rating) VALUES ('Dodge', 'd');
   INSERT INTO Dealer(name, rating) VALUES ('Eagle', 'e');
   INSERT INTO Dealer(name, rating) VALUES ('Ford', 'f');
   INSERT INTO Dealer(name, rating) VALUES ('Geo', 'g');
   INSERT INTO Dealer(name, rating) VALUES ('Honda', 'h');
   INSERT INTO Dealer(name, rating) VALUES ('Isuzu', 'I');
   COMMIT;
   ```

6. Run the following SQL script to create the MobiLink system tables and stored procedures. Replace *C:\Program Files\SQL Anywhere 12* with the location of your SQL Anywhere 12 installation.

   ```
   READ "C:\Program Files\SQL Anywhere 12\MobiLink\setup\syncsa.sql"
   ```

7. Run the following SQL script to specify a download_cursor synchronization script and record the synchronization state to the ml_sis_sync_state system table:

```
CALL ml_add_table_script(
    'CarDealer',
    'Dealer',
    'download_cursor',
    'SELECT * FROM Dealer WHERE last_modified >= ?'
);

CALL ml_add_connection_script(
    'CarDealer',
    'publication_nonblocking_download_ack',
    'CALL ml_set_sis_sync_state(
        {ml s.remote_id},
        NULL,
        {ml s.publication_name},
        {ml s.username},
        NULL,
        {ml s.last_publication_download}
    )'
);

CALL ml_add_table_script(
    'CarDealer', 'Dealer', 'download_delete_cursor', '--{ml_ignore}'
);

COMMIT;
```

This script sets the ml_sis_sync_state to record download-only synchronization. Recording the synchronization state allows you to reference the ml_sis_sync_state system table from the request_cursor event. You specify the request_cursor event in the next lesson.

8. Close Interactive SQL.

See also

- "The SQL Anywhere database server" [*SQL Anywhere Server - Database Administration*]
- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "Writing synchronization scripts" [*MobiLink - Server Administration*]
- "download_cursor table event" [*MobiLink - Server Administration*]
- "publication_nonblocking_download_ack connection event" [*MobiLink - Server Administration*]

Lesson 3: Configure the Notifier

In this lesson, you configure a Notifier event to define how the Notifier creates push requests, and sends push notifications to devices.

Connect to the consolidated database by creating a new MobiLink project.

To create a new MobiLink project

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. From the **Tools** menu, choose **MobiLink 12** » **New Project**.
3. In the **Name** field, type **SIS_CarDealer_LP_DBLSN_CONDB_project**.

4. In the **Location** field, type *C:\MLsis*, and then click **Next**.
5. Check the **Add A Consolidated Database To The Project** option.
6. In the **Database Display Name** field, type **SIS_CarDealer_LP_DBLSN_CONDB**.
7. Click **Edit**. The **Connect To A Generic ODBC Database** window appears.
8. In the **User ID** field, type **DBA**.
9. In the **Password** field, type **sql**.
10. In the **ODBC Data Source** name field, click **Browse** and select **SIS_CarDealer_LP_DBLSN_CONDB**.
11. Click **OK**, then click **Save**.
12. Check the **Remember The Password** option, and then click **Finish**.
13. Click **OK**.

To create a new Notifier

1. In the left pane of Sybase Central under **MobiLink 12**, expand **SIS_CarDealer_LP_DBLSN_CONDB_project**, **Consolidated Databases** and then **SIS_CarDealer_LP_DBLSN_CONDB - DBA**.
2. Right-click **Notification** and choose **New** » **Notifier**.
3. In the **What Do You Want To Name The New notifier** field, type **CarDealerNotifier**.
4. Click **Finish**.

The request_cursor event script detects push requests. Each push request determines what information is sent, and which device should receive the information.

To specify a request_cursor event script

1. In the right pane, select **CarDealerNotifier** and, from the **File** menu, choose **Properties**.
2. Click the **Events** tab and choose **request_cursor** from the **Events** list.
3. Type the following SQL script in the provided text field:

```
SELECT ml_sis_sync_state.remote_id + '.sync' FROM ml_sis_sync_state
WHERE
(
    EXISTS (SELECT 1 FROM Dealer
        WHERE last_modified >= ml_sis_sync_state.last_download)
    AND EXISTS (SELECT poll_key FROM non_sync_request)
)
```

4. Click **OK** to save the Notifier event.

See also

- “request_cursor event” on page 33
- “ml_set_sis_sync_state system procedure” on page 81

Lesson 4: Start the MobiLink server

In this lesson, you start the MobiLink server with the Notifier so that push notifications can be sent to devices.

To run the MobiLink server (mlsrv12)

- Connect to your consolidated database.

  Run the following command:

  ```
  mlsrv12 -notifier -c "dsn=SIS_CarDealer_LP_DBLSN_CONDB" -o serverOut.txt -
  v+ -dl -zu+ -x tcpip
  ```

  The MobiLink server messages window appears. The Notifier indicates that it is ready to receive push notification requests from devices.

The following table describes the mlsrv12 options used in this lesson. The options -o and -v provide debugging and troubleshooting information. Using these logging options is appropriate in a development environment. For performance reasons, -v is typically not used in production.

Option	Description
-notifier	Starts all enabled Notifiers for server-initiated synchronization. See “-notifier mlsrv12 option” [*MobiLink - Server Administration*].
-c	Specifies a connection string. See “-c mlsrv12 option” [*MobiLink - Server Administration*].
-o	Specifies the message log file *serverOut.txt*. See “-o mlsrv12 option” [*MobiLink - Server Administration*].
-v+	Specifies what information is logged. Using -v+ sets maximum verbose logging. See “-v mlsrv12 option” [*MobiLink - Server Administration*].
-zu+	Adds new users automatically. See “-zu mlsrv12 option” [*MobiLink - Server Administration*].
-x	Sets the communications protocol and protocol options for MobiLink clients. See “-x mlsrv12 option” [*MobiLink - Server Administration*].

See also

For more information about the topics in this lesson, see:

- "MobiLink server" [*MobiLink - Server Administration*]
- "MobiLink server options" [*MobiLink - Server Administration*]

Lesson 5: Set up a remote database

In this lesson, you create a SQL Anywhere remote database, create a synchronization publication, a user, and a subscription. You also create the command file for the MobiLink Listener, and then start the MobiLink Listener.

To set up your MobiLink client database

1. Create your MobiLink client database using the dbinit command line utility.

   Run the following command:

   ```
   dbinit SIS_CarDealer_LP_DBLSN_REM
   ```

2. Start your MobiLink client database using the dbeng12 command line utility.

   Run the following command:

   ```
   dbeng12 SIS_CarDealer_LP_DBLSN_REM
   ```

3. Connect to your MobiLink client database using Interactive SQL.

   Run the following command:

   ```
   dbisql -c "server=SIS_CarDealer_LP_DBLSN_REM;uid=DBA;pwd=sql"
   ```

4. Create the Dealer table.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE TABLE Dealer (
       name           VARCHAR(10) NOT NULL PRIMARY KEY,
       rating         VARCHAR(5),
       last_modified  TIMESTAMP DEFAULT TIMESTAMP
   )
   COMMIT;
   ```

5. Create your MobiLink synchronization user, publication, and subscription.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE PUBLICATION CarDealer(TABLE DEALER WHERE 0=1)
   CREATE SYNCHRONIZATION USER test_mluser OPTION ScriptVersion='CarDealer'
   CREATE SYNCHRONIZATION SUBSCRIPTION TO CarDealer FOR test_mluser
   SET OPTION public.ml_remote_id = remote_id;
   COMMIT;
   ```

See also

- "MobiLink clients" [*MobiLink - Client Administration*]
- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "Publishing data" [*MobiLink - Client Administration*]
- "CREATE PUBLICATION statement [MobiLink] [SQL Remote]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE SYNCHRONIZATION USER statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "Script versions" [*MobiLink - Server Administration*]

Lesson 6: Configure the MobiLink Listener

In this lesson, you configure the MobiLink Listener by storing the MobiLink Listener options in a text file, and then running dblsn with the file name specified at the command line.

To configure the MobiLink Listener

1. Run the following command to synchronize with the MobiLink server and create the *SIS_CarDealer_LP_DBLSN_REM.rid* file:

   ```
   dbmlsync -c "server=SIS_CarDealer_LP_DBLSN_REM;uid=DBA;pwd=sql" -e sa=on -
   o rem1.txt -v+
   ```

   The MobiLink Listener can use the $remote_id action variable to define a poll key, which the MobiLink server uses to identify the device. This variable is retrieved from the remote ID file, *SIS_CarDealer_LP_DBLSN_REM.rid*, which is created during the initial synchronization with the MobiLink server. You must synchronize with the MobiLink server to make use of the remote ID file.

2. Click **Shut Down** on the SQL Anywhere MobiLink client window.

3. Create a new text file with the following contents:

   ```
   # Verbosity level
   -v2

   # Show notification messages in console and log
   -m

   # Truncate, then write output to dblsn.txt
   -ot dblsn.txt

   # Remote ID file (defining the scope of $remote_id)
   -r SIS_CarDealer_LP_DBLSN_REM.rid

   # Message handlers

   # Watch for a notification without action
   -l "poll_connect='tcpip(host=localhost)';
       poll_notifier=CarDealerNotifier;
       poll_key=$remote_id.no_action;"

   # Signal dbmlsync to launch, sync and then shutdown
   ```



```
-l "poll_connect='tcpip(host=localhost)';
    poll_notifier=CarDealerNotifier;
    poll_key=$remote_id.sync;
    action='run dbmlsync.exe -c
server=SIS_CarDealer_LP_DBLSN_REM;uid=DBA;pwd=sql -e sa=on -o rem1.txt -v
+';"

# Shutdown the MobiLink Listener
-l "poll_connect='tcpip(host=localhost)';
    poll_notifier=CarDealerNotifier;
    poll_key=$remote_id.shutdown;
    action='DBLSN FULL SHUTDOWN';"
```

4. This tutorial assumes *c:\MLsis* as the working directory for server-side components. Save the text file as *mydblsn.txt* in this directory.

5. Start the MobiLink Listener.

   At a command prompt, navigate to the directory of your MobiLink Listener command file.

   Start the MobiLink Listener by entering:

   ```
   dblsn @mydblsn.txt
   ```

   The **MobiLink Listener for Windows** window appears, indicating the MobiLink Listener is sleeping.

See also

- “Listeners” on page 12
- “MobiLink Listener utility for Windows devices (dblsn)” on page 49
- “@ dblsn option” on page 52
- “Action variables” on page 15

Lesson 7: Issue push requests

In this lesson, you make a change to the Dealer table in the consolidated database so that the information can be downloaded into the remote database when the MobiLink Listener polls for push notifications. You then prompt a server-initiated synchronization by inserting a poll key value into the consolidated database. The Notifier runs the request_cursor event, detects the poll key in the non_sync_request table, then sends a push notification to the MobiLink Listener. When the MobiLink Listener receives the push notification, it synchronizes with the MobiLink database and updates the remote database.

To make a change in the consolidated database

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=SIS_CarDealer_LP_DBLSN_CONDB"
   ```

2. Run the following SQL script:

```
UPDATE Dealer
    SET RATING = 'B' WHERE name = 'Geo';
COMMIT;
```

You issue push requests by populating the non_sync_request table directly. The poll key column determines which device should receive push notifications.

To prompt server-initiated synchronization

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=SIS_CarDealer_LP_DBLSN_CONDB"
   ```

2. Type the following script:

   ```
   INSERT INTO non_sync_request(poll_key) VALUES ('%remote_id%.no_action');
   COMMIT;
   ```

3. Wait a few seconds for the synchronization to occur.

   The MobiLink Listener should poll the consolidated database, download the push notification, then update the Dealer table on the remote database.

To stop server-initiated synchronization with a device, delete the poll key value from the non_sync_request table.

To stop server-initiated synchronization

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=SIS_CarDealer_LP_DBLSN_CONDB"
   ```

2. Type the following script:

   ```
   DELETE FROM non_sync_request WHERE poll_key = '%remote_id%.no_action';
   COMMIT;
   ```

See also

- “Generating push requests” on page 7
- “INSERT statement” [*SQL Anywhere Server - SQL Reference*]
- “UPDATE statement” [*SQL Anywhere Server - SQL Reference*]
- “DELETE statement” [*SQL Anywhere Server - SQL Reference*]

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Close Interactive SQL.
2. Close the SQL Anywhere, MobiLink, and synchronization client windows.
3. Delete all tutorial-related DSNs.
   a. Start the ODBC Data Source Administrator.

      At a command prompt, type the following command:

      ```
      odbcad32
      ```

   b. Remove the **SIS_CarDealer_LP_DBLSN_CONDB** data source.
4. Navigate to the directory containing your consolidated and remote databases, *c:\MLsis*, and delete all the files.

Tutorial: Server-initiated synchronization using gateways

This tutorial demonstrates how to configure a SQL Anywhere consolidated and remote database for server-initiated synchronization. It is based on the sample code located in *samples-dir\MobiLink\SIS_CarDealer*.

Several sample implementations of server-initiated synchronization are located in *samples-dir\MobiLink*. All server-initiated synchronization sample directory names begin with the SIS_ prefix.

Required software

- SQL Anywhere 12

Competencies and experience

- Basic knowledge of MobiLink event scripts.

Goals

- Set up a SQL Anywhere consolidated database for server-initiated synchronization.
- Configure server-side properties.
- Issue push requests to prompt a server-initiated synchronization.

Suggested background reading

- “Introduction to server-initiated synchronization” on page 1

Lesson 1: Set up the consolidated database

In this lesson, you create a consolidated database named **MLConsolidated** with the scripts required for synchronization using the dbinit utility.

To create and start a new SQL Anywhere consolidated database

1. Create a new working directory to store the consolidated database.

   This tutorial assumes *c:\MLsis* as the working directory.

2. Create the SQL Anywhere consolidated database using the dbinit utility.

3. Run the following command:

   ```
   dbinit MLconsolidated
   ```

4. Start the consolidated database using the dbeng12 utility.

   Run the following command:

   ```
   dbeng12 MLconsolidated
   ```

Define an ODBC data source for the consolidated database

Use the SQL Anywhere 12 driver to define an ODBC data source for the database.

To define an ODBC data source for the consolidated database

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **ODBC Data Source Administrator**.

2. Click the **User DSN** tab, and click **Add**.

3. In the **Create New Data Source** window, click **SQL Anywhere 12** and click **Finish**.

4. Perform the following tasks in the **ODBC Configuration For SQL Anywhere** window:
   a. Click the **ODBC** tab.
   b. In the **Data Source Name** field, type **sis_cons**.
   c. Click the **Login** tab.
   d. In the **User ID** field, type **DBA**.
   e. In the **Password** field, type **sql**.
   f. From the **Action** dropdown list, choose **Connect To A Running Database On This Computer**.
   g. In the **Server Name** field, type **MLconsolidated**.
   h. Click **OK**.

5. Close the ODBC data source administrator.

Click **OK** on the **ODBC Data Source Administrator** window.

See also

- "Creating ODBC data sources" [*SQL Anywhere Server - Database Administration*]

Lesson 2: Generate database schema

In this lesson, you generate the database schema, which includes a Dealer table and a download_cursor synchronization script. A table and stored procedure is used to generate server-initiated synchronization push requests.

To set up the database schema

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. Perform the following tasks to connect to the consolidated database:
   a. From the **Connections** menu, choose **Connect with SQL Anywhere 12**.
   b. From the **Action** dropdown list, choose **Connect with an ODBC Data Source**.
   c. Click **ODBC Data Source Name**, and click **Browse**.
   d. Select **sis_cons** and then click **OK**.
   e. Click **Connect**.
3. Connect to your database using Interactive SQL.

   You can start Interactive SQL from Sybase Central or from a command prompt.

   - To start Interactive SQL from Sybase Central, right-click the **MLconsolidated - DBA** database and click **Open Interactive SQL**.
   - To start Interactive SQL at a command prompt, run the following command:

     ```
     dbisql -c "dsn=sis_cons"
     ```
4. Run the following SQL script to create and set up the Dealer table:

   ```
   CREATE TABLE Dealer (
       name VARCHAR(10) NOT NULL PRIMARY KEY,
       rating VARCHAR(5),
       last_modified TIMESTAMP DEFAULT TIMESTAMP
   )
   ```
5. Insert data into the Dealer table using the following statements:

   ```
   INSERT INTO Dealer(name, rating) VALUES ('Audi', 'a');
   INSERT INTO Dealer(name, rating) VALUES ('Buick', 'b');
   INSERT INTO Dealer(name, rating) VALUES ('Chrysler', 'c');
   INSERT INTO Dealer(name, rating) VALUES ('Dodge', 'd');
   INSERT INTO Dealer(name, rating) VALUES ('Eagle', 'e');
   INSERT INTO Dealer(name, rating) VALUES ('Ford', 'f');
   INSERT INTO Dealer(name, rating) VALUES ('Geo', 'g');
   INSERT INTO Dealer(name, rating) VALUES ('Honda', 'h');
   ```

```
INSERT INTO Dealer(name, rating) VALUES ('Isuzu', 'I');
COMMIT;
```

6. Run the following SQL script to create the MobiLink system tables and stored procedures. Replace *C:\Program Files\SQL Anywhere 12* with the location of your SQL Anywhere 12 installation.

```
READ "C:\Program Files\SQL Anywhere 12\MobiLink\setup\syncsa.sql"
```

7. Run the following SQL script to specify a download_cursor synchronization script and record the synchronization:

```
CALL ml_add_table_script(
    'sis_ver1',
    'Dealer',
    'download_cursor',
    'SELECT * FROM Dealer WHERE last_modified >= ?'
);

CALL ml_add_table_script(
    'sis_ver1', 'Dealer', 'download_delete_cursor', '--{ml_ignore}'
);

COMMIT
```

See also

- "The SQL Anywhere database server" [*SQL Anywhere Server - Database Administration*]
- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "Writing synchronization scripts" [*MobiLink - Server Administration*]
- "download_cursor table event" [*MobiLink - Server Administration*]

Lesson 3: Create a table to store push request

In this lesson, you create a push request table for storing push requests. The Notifier sends a message to a device when it detects a push request.

To create a simple table for storing push requests

1. Connect to your database using Interactive SQL.

   You can start Interactive SQL from Sybase Central or from a command prompt.

   - To start Interactive SQL from Sybase Central, right-click the **sis_cons - DBA** database and click **Open Interactive SQL**.
   - To start Interactive SQL at a command prompt, run the following command:

     ```
     dbisql -c "dsn=sis_cons"
     ```

2. Run the following SQL script in Interactive SQL:

```
CREATE TABLE PushRequest (
    req_id INTEGER DEFAULT AUTOINCREMENT PRIMARY KEY,
    mluser VARCHAR(128),
    subject VARCHAR(128),
    content VARCHAR(128),
```



```
        resend_interval VARCHAR(30) DEFAULT '20s',
        time_to_live VARCHAR(30) DEFAULT '1m',
        status VARCHAR(128) DEFAULT 'created'
    )
    COMMIT;
```

3. Close Interactive SQL.

See also

- "Push requests" on page 5
- "Introduction to server-initiated synchronization" on page 1
- "Server-initiated synchronization components" on page 2

Lesson 4: Configure the Notifier

In this lesson, you configure three Notifier events to define how the Notifier creates push requests, transmits the requests to MobiLink Listeners, and deletes expired requests.

Connect to the consolidated database by creating a new MobiLink project.

To create a new MobiLink project

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. From the **Tools** menu, choose **MobiLink 12** » **New Project**.
3. In the **Name** field, type **sis_cons_project**.
4. In the **Location** field, type *C:\MLsis*, and then click **Next**.
5. Check the **Add A Consolidated Database To The Project** option.
6. In the **Database Display Name** field, type **sis_cons**.
7. Click **Edit**. The **Connect To A Generic ODBC Database** window appears.
8. In the **User ID** field, type **DBA**.
9. In the **Password** field, type **sql**.
10. In the **ODBC Data Source** name field, click **Browse** and select **sis_cons**.
11. Click **OK**, then click **Save**.
12. Check the **Remember The Password** option, and then click **Finish**.
13. Click **OK**.

To create a new Notifier

1. In the left pane of Sybase Central under **MobiLink 12**, expand **sis_cons_project**, **Consolidated Databases** and then **sis_cons**.
2. Right-click **Notification** and choose **New** » **Notifier**.
3. In the **What Do You Want To Name The New notifier** field, type **CarDealerNotifier**.
4. Click **Finish**.

The Notifier detects changes in the consolidated database and creates push requests using the begin_poll event. In this case, the begin_poll script populates the PushRequest table if changes occur in the Dealer table and when a remote database is not up-to-date.

The request_cursor script fetches push requests. Each push request determines what information is sent in the message, and which remote databases receive the information.

The request_delete Notifier event specifies cleanup operations. Using this script, the Notifier can automatically remove implicitly dropped and expired requests.

To configure the Notifier properties and events

1. Enter the begin_poll event script.
   a. In the right pane, select **CarDealerNotifier** and, from the **File** menu, choose **Properties**.
   b. Click the **Events** tab.
   c. Choose **begin_poll** from the **Events** list.
   d. Type the following SQL script in the provided text field:

```
--
-- Insert the last consolidated database
-- modification date into @last_modified
--
DECLARE @last_modified timestamp;
SELECT MAX(last_modified) INTO @last_modified FROM Dealer;

--
-- Delete processed requests if the mluser is up-to-date
--
DELETE FROM PushRequest
    FROM PushRequest AS p, ml_user AS u, ml_subscription AS s
    WHERE p.status = 'processed'
        AND u.name = p.mluser
        AND u.user_id = s.user_id
        AND @last_modified <= GREATER(s.last_upload_time,
s.last_download_time);

--
-- Insert new requests when a device is not up-to-date
--
INSERT INTO PushRequest(mluser, subject, content)
SELECT u.name, 'sync', 'ignored'
    FROM ml_user as u, ml_subscription as s
    WHERE u.name IN (SELECT name FROM ml_listening WHERE listening =
```



```
'y')
        AND u.user_id = s.user_id
        AND @last_modified > greater(s.last_upload_time,
s.last_download_time)
        AND u.name NOT LIKE '%-dblsn'
        AND NOT EXISTS(SELECT * FROM PushRequest
            WHERE PushRequest.mluser = u.name
                AND PushRequest.subject = 'sync')
```

In the first major section of the begin_poll script, processed requests from the PushRequest table are eliminated if a device is up to date:

```
@last_modified <= GREATER(s.last_upload_time, s.last_download_time)
```

@last_modified is the maximum modification date in the consolidated database Dealer table. The expression greater(s.last_upload_time, s.last_download_time) represents the last synchronization time for a remote database.

You can also delete push requests directly using the request_delete event. However, the begin_poll event, in this case, ensures that expired or implicitly dropped requests are not eliminated before a remote database synchronizes.

The next section of code checks for changes in the last_modified column of the Dealer table and issues push requests for all active MobiLink Listeners (listed in the ml_listening table) that are not up to date:

```
@last_modified > GREATER(s.last_upload_time, s.last_download_time)
```

When populating the PushRequest table, the begin_poll script sets the subject to 'sync'.

2. Enter the request_cursor script.
   a. Choose **request_cursor** from the **Events** list.
   b. Type the following SQL script in the provided text field:

```
SELECT
    p.req_id,
    'Default-DeviceTracker',
    p.subject,
    p.content,
    p.mluser,
    p.resend_interval,
    p.time_to_live
    FROM PushRequest AS p
```

The PushRequest table supplies rows to the request_cursor script.

The order and values in the request_cursor result set is significant. The second parameter, for example, defines the default gateway Default-DeviceTracker. A device tracking gateway keeps track of how to reach users and automatically selects UDP or SMTP to connect to remote devices.

3. Enter the request_delete script.
   a. Choose **request_delete** from the **Events** list.
   b. Type the following SQL script in the provided text field:

```
UPDATE PushRequest SET status='processed' WHERE req_id = ?
```

Instead of deleting the row, this request_delete script updates the status of a row in the PushRequest table to 'processed'.

4. Click **OK** to save the Notifier events.

See also

- "request_delete event" on page 34
- "begin_poll event" on page 30
- "Device tracking gateways" on page 20
- "request_cursor event" on page 33
- "request_delete event" on page 34

Lesson 5: Configure gateways and carriers

Gateways are the mechanisms for sending messages. You can define either a supported gateway or a device tracking gateway. The MobiLink server keeps track of how to reach clients when you specify a device tracking gateway, and automatically chooses the most appropriate gateway.

You use a default device tracking gateway for the purpose of this tutorial, so configuration is not necessary.

See also

- "Using gateways as an alternative to light weight pollers" on page 19
- "Gateways and carriers" on page 19
- "Device tracking gateway properties" on page 43

Lesson 6: Start the MobiLink server

In this lesson, you start the MobiLink server with the Notifier so that push notifications can be sent to devices.

To run the MobiLink server (mlsrv12)

- Connect to your consolidated database.

  Run the following command:

  ```
  mlsrv12 -notifier -c "dsn=sis_cons" -o serverOut.txt -v+ -dl -zu+ -x tcpip
  ```

  The MobiLink server messages window appears. The Notifier indicates that it is ready to receive push notification requests from devices.

The following table describes the mlsrv12 options used in this lesson. The options -o and -v provide debugging and troubleshooting information. Using these logging options is appropriate in a development environment. For performance reasons, -v is typically not used in production.

Option	Description
-notifier	Starts all enabled Notifiers for server-initiated synchronization. See "-notifier mlsrv12 option" [*MobiLink - Server Administration*].
-c	Specifies a connection string. See "-c mlsrv12 option" [*MobiLink - Server Administration*].
-o	Specifies the message log file *serverOut.txt*. See "-o mlsrv12 option" [*MobiLink - Server Administration*].
-v+	Specifies what information is logged. Using -v+ sets maximum verbose logging. See "-v mlsrv12 option" [*MobiLink - Server Administration*].
-zu+	Adds new users automatically. See "-zu mlsrv12 option" [*MobiLink - Server Administration*].
-x	Sets the communications protocol and protocol options for MobiLink clients. See "-x mlsrv12 option" [*MobiLink - Server Administration*].

See also

For more information about the topics in this lesson, see:

- "MobiLink server" [*MobiLink - Server Administration*]
- "MobiLink server options" [*MobiLink - Server Administration*]

Lesson 7: Set up a remote database

In this lesson, you create a SQL Anywhere remote database, create a synchronization publication, a user, and a subscription. You also create the command file for the MobiLink Listener, and then start the MobiLink Listener.

To set up your MobiLink client database

1. Create your MobiLink client database using the dbinit command line utility.

   Run the following command:

   ```
   dbinit remote1
   ```

2. Start your MobiLink client database using the dbeng12 command line utility.

   Run the following command:

```
dbeng12 remote1
```

3. Connect to your MobiLink client database using Interactive SQL.

   Run the following command:

   ```
   dbisql -c "server=remote1;uid=DBA;pwd=sql"
   ```

4. Create the Dealer table.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE TABLE Dealer (
       name            VARCHAR(10) NOT NULL PRIMARY KEY,
       rating          VARCHAR(5),
       last_modified   TIMESTAMP DEFAULT TIMESTAMP
   )
   COMMIT;
   ```

5. Create your MobiLink synchronization user, publication, and subscription.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE PUBLICATION car_dealer_pub (table Dealer);
   CREATE SYNCHRONIZATION USER sis_user1;
   CREATE SYNCHRONIZATION SUBSCRIPTION
       TO car_dealer_pub
       FOR sis_user1
       OPTION scriptversion='sis_ver1';
   COMMIT;
   ```

See also

- "MobiLink clients" [*MobiLink - Client Administration*]
- "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*]
- "Publishing data" [*MobiLink - Client Administration*]
- "CREATE PUBLICATION statement [MobiLink] [SQL Remote]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE SYNCHRONIZATION USER statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "Script versions" [*MobiLink - Server Administration*]

Lesson 8: Configure the MobiLink Listener

In this lesson, you configure the MobiLink Listener by storing the MobiLink Listener options in a text file, and then running dblsn with the file name specified at the command line.

To configure the MobiLink Listener

1. Create a new text file with the following contents:

   ```
   #---------------------------------
   # Verbosity level
   ```



```
-v2

# Show notification messages in console and log
-m

# Polling interval, in seconds
-i 3

# Truncate, then write output to dblsn.txt
-ot dblsn.txt

# MobiLink address and connect parameter for dblsn
-x "host=localhost"

# Enable device tracking and specify the MobiLink user name.
-t+ sis_user1

# Message handlers
# Synchronize using dbmlsync
-l "subject=sync;
action='start dbmlsync.exe
 -c server=remote1;uid=DBA;pwd=sql
 -o dbmlsyncOut.txt';"
```

2. This tutorial assumes *c:\MLsis* as the working directory for server-side components. Save the text file as *mydblsn.txt* in this directory.

3. Start the MobiLink Listener.

   At a command prompt, navigate to the directory of your MobiLink Listener command file.

   Start the MobiLink Listener by entering:

   ```
   dblsn @mydblsn.txt
   ```

   The **MobiLink Listener for Windows** window appears, indicating the MobiLink Listener is sleeping.

When tracking information is uploaded to the consolidated database, a new entry appears in the MobiLink server messages window. This information relays the successful initial communication between the MobiLink Listener and the MobiLink server.

See also

- "Listeners" on page 12
- "MobiLink Listener utility for Windows devices (dblsn)" on page 49
- "@ dblsn option" on page 52

Lesson 9: Issue push requests

For server-initiated synchronization, you can issue push requests by populating the PushRequest table directly, or making a change in the Dealer table. In the latter case, the Notifier begin_poll script detects the change in the Dealer table and populate the PushRequest table.

In either case, the PushRequest table supplies rows to the Notifier request_cursor script, which determines how remote devices receive messages.

To insert a push request directly into the PushRequest table prompting server-initiated synchronization

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=sis_cons"
   ```

2. Run the following SQL script:

   ```
   INSERT INTO PushRequest(mluser, subject, content)
       VALUES ('sis_user1', 'sync', 'not used');
   COMMIT;
   ```

3. Wait a few seconds for the synchronization to occur.

When populated, the PushRequest table supplies rows to the Notifier's request_cursor script. The request_cursor script determines what information is sent in the message, and which remote devices receive the information.

To make a change in the consolidated database Dealer prompting server-initiated synchronization

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=sis_cons"
   ```

2. Run the following SQL script:

   ```
   UPDATE Dealer
       SET RATING = 'B' WHERE name = 'Geo';
   COMMIT;
   ```

3. Wait a few seconds for the synchronization to occur.

In this case, the Notifier begin_poll script detects changes in the dealer table and populates the PushRequest table appropriately. As before, once the PushRequest table is populated, the Notifier request_cursor script determines what information is sent in the message, and which remote devices receive the information.

See also

- “Generating push requests” on page 7
- “INSERT statement” [*SQL Anywhere Server - SQL Reference*]
- “UPDATE statement” [*SQL Anywhere Server - SQL Reference*]

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Close Interactive SQL.
2. Close the SQL Anywhere, MobiLink, and synchronization client windows.
3. Delete all tutorial-related DSNs.
   a. Start the ODBC Data Source Administrator.

      At a command prompt, type the following command:

      ```
      odbcad32
      ```

   b. Remove the **sis_cons** data source.
4. Navigate to the directory containing your consolidated and remote databases, *c:\MLsis*, and delete all the files.

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

U

W

