



MoneyScape Server Administrator Guide

MoneyScape Version 1.0

Introduction

Who Should Read This Book	1
How This Book Is Organized	1
Conventions Used in This Book	1
Technical Support	1
Related Publications	1

Getting Started

Technical Specifications	4
Hardware Requirements	4
Software Requirements	5
Before You Begin	5
Getting a Transmitter License	5
Getting a Digital Certificate	5
Configuring for Oracle7	6
Configuring for Oracle8	7
Creating a Unix User Account	8
Optional Solaris Tuning	9

Installing Server Software

Getting Started	12
Configuring Transmitter Security	12
Choosing Installation Directories	12
Installing the Server	13
Installation Dialog	14
What Gets Installed	16
Uninstalling the Server	17
Backing Up the Server	18
Installing a Transmitter License	18

Installing a Digital Certificate	18
Importing a Certificate	19
Starting the Server and Transmitter	19
Stopping the Server and Transmitter	20
Installing the Developer Workstation	20
Windows Installation	20
Solaris Installation	21
Installing the Tools	21
Installing the System and Sample Channels	22
Installing the Documentation	22
Installing the Client Files (Windows only)	23
Setting the Default Transmitter	23
Publishing the System and Sample Channels	24
Selecting a Startup Channel	24
Verifying the Installation	25
Installing the Sample Client	25
Setting the Default Transmitter	26
Running the Sample Client	26

Using the Server Administrator

How the Server Administrator Works	27
Adapters, Datafeeds, and Channels	28
Authentication Server	28
Admin Properties	29
Logging In	30
Using the Server Administrator	30
Starting and Stopping a Transmitter	32

Configuring Channels and Datafeeds

Configuration Overview	35
How Adapters Work	36
Data That Adapters Retrieve	36
Creating Channel Records	37
Adding, Modifying, and Deleting Records	38
Creating Datafeed Records	42
Configuration Examples	46
AnswerMyQuestion Channel and Adapter – Simple Example	47
AnswerMyQuestion Channel and Adapter – Complex Example	48
AnswerMyQuestion – Using Cache	51

Frequently Asked Questions

Trademarks

MECA Software L.L.C. is a registered trademark and MoneyScape is a trademark in the United States and other countries.

Adobe, Acrobat, and PostScript are trademarks of Adobe Systems, Inc. Marimba and Castanet are registered trademarks, trademarks, or service marks of Marimba, Inc. Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation. Oracle and Oracle7 are trademarks of Oracle Corp. Sun, Sun Microsystems, Solaris, Java, and Javasoft are trademarks or registered trademarks of Sun Microsystems, Inc. VeriSign is a trademark of VeriSign, Inc. Unix is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

All other products or services mentioned in this document are identified by the trademarks, service marks, or product names as designated by the companies who market those products. Inquiries concerning such trademarks should be made directly to those companies.

Copyright



© 1999 MECA Software L.L.C. All rights reserved. 115 Corporate Drive, Trumbull, Connecticut 06611 USA

This publication and the software it describes contain proprietary and confidential information. No part of this document may be copied, photocopied, reproduced, translated, or reduced to any electronic or machine-readable format without prior written permission of MECA Software.

The information in this document is subject to change without notice. MECA Software assumes no responsibility for any damages arising from the use of this document, including but not limited to, lost revenue, lost data, claims by third parties, or other damages. If you have comments or suggestions concerning this manual, please contact documentation@mecasw.com.

Introduction

Who Should Read This Book

This guide is written for Solaris system administrators. It explains all administrative tasks for MoneyScape servers. It assumes that you are familiar with the Java development environment, object-oriented programming, and with MoneyScape concepts and facilities (see [Related Publications](#)).

How This Book Is Organized

- [Getting Started](#) contains a description of the server as well as installation requirements and technical specifications.
- [Installing Server Software](#) explains how to install all server-related software including digital certificates, and MoneyScape tools and documentation.
- [Using the Server Administrator](#) explains how to use the Server Administrator tool to manage the server, start/stop transmitters, etc.
- [Configuring Channels and Datafeeds](#) explains how to add and configure datafeeds and channels so that they can communicate and pass information.
- [Frequently Asked Questions](#) provides a list of answers to commonly asked questions.

Conventions Used in This Book

By convention, this Courier typewriter font is used to indicate Java classes, methods, and other code-related elements.

Technical Support

MECA Software is committed to providing the service and support needed to build, install, and manage your MoneyScape environment. For information about service programs, or for technical support or training, contact [MECA](#) online or call your account representative.

Related Publications

In addition to this *Server Administrator Guide*, you may want to refer to the following MoneyScape documents for more information. These publications are available on the installation CD-ROM in HTML or Adobe PDF format.

- *Channel Developer Guide* – explains all aspects of channel development, including how to write and publish Java channels and HTML channels, how to write adapters that interface with external data sources, and how to use the MoneyScape tools.

- *Concepts and Facilities Guide* – describes the conceptual framework behind MoneyScape and explains how to successfully exploit key features.
- *Client Customization Guide* – explains how to brand and customize the MoneyScape client application.

Getting Started

The MoneyScape server is designed to provide configurable, targeted content to customers through channels to which they subscribe. It is designed to integrate into existing environments, providing common interface hookpoints that easily let you present existing data and functionality to your customers. While the server remains running, and without interruption or any delay in responding to client requests, MoneyScape lets you

- manage the content for channels.
- add or remove channels.
- temporarily disable entire channels or subsets of the data provided by channels.
- add new adapters, the customized Java modules that give you a hookpoint to integrate existing data and functionality into MoneyScape.

It is designed to run 24 hours a day, seven days a week. The MoneyScape server is used to start/stop/manage transmitters, add/delete/modify content, and perform other administrative tasks. The transmitter portion of the server is contacted by the MoneyScape client application for subscribing/unsubscribing, receiving generic channel data, as well as for user-specific information. The transmitter can be set to run in SSL or non-SSL mode. SSL (Secure Sockets Layer) is a leading security protocol on the Internet that lets the browser and server exchange data via secret key encryption.

MoneyScape systems administrators use the Server Administrator tool to contact and manage MoneyScape servers. The Server Administrator is itself a client/server application and the client application can run on machines other than the host where MoneyScape is installed. The Server Administrator client can run locally or remotely on Solaris or Windows 95/NT machines. *One Server Administrator client is needed for each MoneyScape server application and it cannot operate from behind a firewall.* Other features include:

- An OFX interface and/or OFX parsing into OFX Beans available to adapter developers.
- An interface for usage tracking and customer profiling.
- Scalability to multiple hosts.
- JDBC support for data storage allowing maximum versatility in your data storage requirements.
- Local data caching to increase performance and reduce connections to outside data sources.

Topics in this section

- [Technical Specifications](#)
- [Before You Begin](#)

Technical Specifications

The MoneyScape server software is written in pure Java with no native components. It is designed and implemented to build and run with the standard JDK packages. Except for the JAR files supplied by MECA and certain embedded components from Oracle and Marimba, no other third-party components are required.

The server consists of a background Java process which is used to manage both data and other background processes (transmitters). The server is managed by a Server Administrator client. The Server Administrator client is also pure Java and communicates with a running server via Remote Method Invocation (RMI). There is authentication for different levels of administrator permissions, and any command is authenticated by the server before execution.

Current implementations of Java do not provide SSL for RMI. For this reason, all startup and runtime properties reside in property files located in an `etc` directory. All data storage is done through JDBC; this lets you control security, backup, and redundancy.

Important The MoneyScape v1.0 server has been tested with an Oracle7 Version 7.3.3 database. There may be conflicts if you install MoneyScape on a server where an older version of Oracle is installed.

Hardware Requirements

- To install a digital certificate using the Marimba utility, you need an X terminal.
- One port is also needed per transmitter through the firewall if there is a requirement for publishing from outside the firewall. This port number is the transmitter port number plus 2000. (*Note that you cannot run the Server Administrator from behind a firewall.*)
- On Solaris servers, the server and transmitters will run as a user. When the installation script runs, it asks this user's name and sets the server to run as that user. This user must exist and have access to the database of choice through JDBC. The server picks up the environment from that user and uses this environment to access other objects.
 - Latest JRE or JDK 1.1.6
 - The host computer must allow a minimum of 1024 file descriptors per process. To modify, add the following lines to `/etc/system`

```
set rlim_fd_cur = 1024
set rlim_fd_max = 1024
```
- The MoneyScape installation requires up to 350 megabytes of disk storage on your server. This includes an Oracle7 server and database that are used exclusively by MoneyScape. As shown below, the disk requirements are substantially less if you already have an Oracle server installed. Also, as you develop channels, more disk space is required in `/export/home`

File System	Required Disk Storage
<code>/opt</code>	20 megabytes (MB)
<code>/export/home</code>	50 MB (without Oracle) 250 MB (with Oracle)
<code>/</code>	10 MB
<code>/tmp (swap)</code>	20 MB

Software Requirements

MoneyScape v1.0 has been tested with the following environment. See the appropriate readme file for updates.

- Sun Solaris 2.6 with latest Sun patch cluster including:
 - Solaris patch 105490-04 (required for JDK)
 - Solaris patch 105284-12
 - Solaris patch 105181-10
- Sun Java JDK 1.1.6 for Solaris (has tested faster than the JavaSoft JVM)

Before You Begin

Before you install MoneyScape, there are several tasks you may need to perform.

- Getting a Transmitter License – You need a special license from MECA for each transmitter. There is generally a 48 hour turnaround for license requests.
- Getting a Digital Certificate – You need a digital certificate only if you are installing a secure transmitter. This generally takes 3–5 days and VeriSign is the only supported vendor.
- Configuring for Oracle7 – You need to set certain Oracle and Solaris parameters depending on whether or not you already have a database installed on your Solaris server.
- Configuring for Oracle8 – MoneyScape is initially configured to run with Oracle7. If you are going to run with Oracle8, you need to make minor adjustments.
- Creating a Unix Account – You need to set up a dedicated operating system user account for each transmitter.
- Optional Solaris Tuning – You may want to set certain Solaris parameters to improve performance. These settings are not required and you may choose not to implement them based on other system requirements.

Getting a Transmitter License

You must get a transmitter license from MECA for each MoneyScape transmitter. Contact your account representative at MECA, before server installation, and you will be e-mailed a license file within 48 hours.

Getting a Digital Certificate

A digital certificate provides encrypted security for communication between client and server. When a transmitter is running in SSL mode, all channels and all background communication between client and server are secured via the SSL protocol. SSL can adversely affect performance so you should plan in advance for channels that will be published with SSL.

Currently, VeriSign, Inc. is the only certificate authority used by Castanet products. A digital certificate remains valid for a limited time and takes 3–5 days to obtain. As a convenience, Marimba and VeriSign have created a test certificate. The test certificate is free and expires two weeks after issue. See the FAQs in this document and contact VeriSign for more information.

In MoneyScape, each authentication server communicates with one transmitter and you can have multiple authentication server/transmitter pairs on a host machine. SSL is enabled on a per

Important A digital certificates provides a secure handshaking mechanism between client and server. A digital certificate does not protect the RMI communication between the authentication server and a transmitter.

transmitter basis but you only need one digital certificate *per host*. **You do not need a separate digital certificate for each secure transmitter.**

When installing a transmitter, you are prompted to enable the transmitter for SSL. If you choose to enable SSL, you must run the Marimba utility that lets you to manage the certificate. *If you configure for SSL, you can't start the transmitter without a digital certificate.* Note that the Marimba utility and the MoneyScape Server Administrator are GUI applications that require an X terminal.

Configuring for Oracle7

Oracle7 Version 7.3.3 is installed as part of the MoneyScape installation if you do not have an existing Oracle7 Server. **Before installing MoneyScape, you must do one of the following depending on whether or not you already have Oracle7 Server installed on your server.**

- *If you don't have Oracle7 Server installed* – you only need to tune the Solaris kernel.
- *If you already have Oracle7 Server installed* – you only need to set the Oracle `processes` variable in the `initXXXX.ora` file.

Tuning the Solaris Kernel

If you don't have Oracle7 database, you must modify the IPC parameters as explained below. The Unix kernel interprocess communication (IPC) parameters need to be configured to accommodate the System Global Area (SGA) structure of the Oracle7 Server. The database will not start if the system doesn't have adequate shared memory for the SGA.

▼ To modify the Solaris server IPC parameters

1. Add the following lines of text to the file `/etc/system`
2. When done, shut down the system in command line mode and reboot with `boot -r`

```
set shmsys:shminfo_shmmax=8388608
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmni=100
set shmsys:shminfo_shmseg=10
set semsys:seminfo_semmns=200
set semsys:seminfo_semmni=70
set semsys:seminfo_semmsl=200
```

Setting the Processes Variable

If you already have Oracle7 Server installed, all variables in the `initXXXX.ora` file (where `XXXX` is the database name) must be tuned to at least `MEDIUM` with the exception of the `processes` variable which must be `LARGE`. The `initXXXX.ora` file is usually in the directory `$ORACLE_HOME/dbs`. Set the variables as shown in the example.

▼ To edit the processes variable

1. Edit the file `initXXXX.ora`

2. Set all variables (except `processes`) to `MEDIUM`.
3. Set `processes = 200` to `LARGE`
4. Reboot when done to restart the database.

```
# tuning parameters

db_files = 20

# db_file_multiblock_read_count = 8           # SMALL
db_file_multiblock_read_count = 16          # MEDIUM
# db_file_multiblock_read_count = 32         # LARGE

# db_block_buffers = 200                     # SMALL
db_block_buffers = 550                      # MEDIUM
# db_block_buffers = 3200                   # LARGE

# shared_pool_size = 3500000                # SMALL
shared_pool_size = 6000000                  # MEDIUM
# shared_pool_size = 9000000                # LARGE

log_checkpoint_interval = 10000

# processes = 50                             # SMALL
# processes = 100                            # MEDIUM
processes = 200                             # LARGE

# dml_locks = 100                           # SMALL
dml_locks = 200                             # MEDIUM
# dml_locks = 500                           # LARGE

# log_buffer = 8192                          # SMALL
log_buffer = 32768                           # MEDIUM
# log_buffer = 163840                        # LARGE

# sequence_cache_entries = 10               # SMALL
sequence_cache_entries = 30                  # MEDIUM
# sequence_cache_entries = 100              # LARGE

# sequence_cache_hash_buckets = 10          # SMALL
sequence_cache_hash_buckets = 23            # MEDIUM
# sequence_cache_hash_buckets = 89          # LARGE

# audit_trail = true                         # if you want auditing
# timed_statistics = true                    # if you want timed statistics
max_dump_file_size = 10240                  # limit trace file size to 5 MB each
```

Configuring for Oracle8

MoneyScape v1.0 is initially configured to run with Oracle7. With minor adjustments it can also run with an existing Oracle8 Version 8.0.4 host installation. Before you begin check that the following tasks have been performed.

- Oracle8 must be installed and running on the host.
- The MoneyScape user that will own and run the MoneyScape server must exist and have access to the Oracle8 database. See [Creating a Unix User Account](#) for details.

- The JDBC drivers and libraries that come with Oracle8 must be installed. If not, install from the CD-ROM (or download the Oracle8 JDBC package from the Oracle web site).

▼ To configure MoneyScape to run with Oracle8

1. Make sure that Oracle8 is running on the host.
2. Install MoneyScape v1.0 with a normal `pkgadd` as explained in the [Installing the Server](#). This process will detect the presence of Oracle8 and will not try to install Oracle7.
3. When the `pkgadd` successfully completes, modify the MoneyScape user's `LD_LIBRARY_PATH` environment variable to include the path `$ORACLE_HOME/lib` so that it will find the file `$ORACLE_HOME/lib/liboci804jdbc.so`
4. The MoneyScape installation creates the file `classes111.zip` in `$HOME/lib` for the MoneyScape user. Do not use this file. Instead, modify the user's `CLASSPATH` as follows. Replace `$HOME/lib/classes111.zip` with `$ORACLE_HOME/jdbc/lib/classes111.zip`
5. Modify the file `$HOME/etc/DataBase.txt` that contains JDBC connection data. Replace all instances of the string `oci7` with the string `oci8` as in the following example.
Change `jdbc:oracle:oci7:dbname.world` to `jdbc:oracle:oci8:dbname.world`
6. Run the MoneyScape server and you are done.

Creating a Unix User Account

Before you can install MoneyScape, you need to create a separate operating system user account on the server for each transmitter that you install. *When creating this account, you can only use the Bourne shell.* This user is called the transmitter user and is referred to in code as `<os_user_name>`.

On Solaris servers, the server and transmitters will run as a user. When the installation script runs, it asks this user's name and sets the server to run as that user. This user must exist and have access to the database of choice. The server picks up the environment from that user and uses this environment to access other objects.

Oracle7 Version 7.3.3 is installed as part of the MoneyScape server installation and the transmitter user must be configured with the JDK and Oracle environments. Installation requirements vary depending on whether or not you already have Oracle7 Server installed on your Solaris server.

- *If you don't have Oracle7 Server installed* – the MoneyScape installation sets all required transmitter user environment variables needed for Oracle and the JDK. The only thing you need to do in advance is create a Unix user account.
- *If you already have Oracle7 Server installed* – the server software must be Oracle7 and the transmitter user (the operating system user account dedicated to the transmitter) must have the environment variables set properly.

The environment variables (shown in bold below) are usually defined in the user's `$HOME/.profile` file. **Note that the following code is only an example.** The values you actually use must be appropriate for your own site.

Sample `$HOME/.profile`

```
$HOME/.profile
```

```
ORACLE_BASE=/opt/u01/app/oracle
ORACLE_HOME=$ORACLE_BASE/product/7.3.3
ORACLE_SID=MECA
export ORACLE_BASE ORACLE_HOME ORACLE_SID
JAVA_HOME=/usr/java
PATH=./usr/java/bin:/opt/bin:/bin:/usr/bin:$ORACLE_HOME/bin:/usr/ccs/bin:/usr/u
cb
export JAVA_HOME PATH
#
# Meca Environment Variables
# Tue Feb 16 10:59:51 EST 1999
#
CLASSPATH="$CLASSPATH:.$HOME/lib/classes111.zip:$HOME/lib/
MecaServer.jar:$HOME/
outside_classes"
LD_LIBRARY_PATH="$HOME/lib"
MECA_JAVAARGS="-DMECA_HOME=$HOME"
export CLASSPATH LD_LIBRARY_PATH MECA_JAVAARGS
```

Optional Solaris Tuning

The following settings are optional and may improve performance. These are usually entered in the `/etc/rc2.d/S69inet` startup script. They are not required and you may choose not to implement them based on other system requirements.

▼ To tune Solaris for optimum performance

1. Add the following lines of text to the file `/etc/rc2.d/S69inet`
2. When done, reboot with `boot -r`

```
ndd -set /dev/tcp tcp_rexmit_interval_min 1500
ndd -set /dev/tcp tcp_close_wait_interval 60000
ndd -set /dev/tcp tcp_slow_start_inital 2
ndd -set /dev/tcp tcp_conn_req_max_q 1024
```

Installing Server Software

The MoneyScape server application runs in a Solaris environment. It manages the transmitter, the datafeeds and adapters, and includes a database for persistent storage. There may be multiple servers but each server is associated with one transmitter. They are a pair.

All server software is installed using the `pkgadd` command on Solaris. The installation requires a Solaris user account that will be the owner of the MoneyScape server and tools; when the server runs, it runs as that user. This user must have the ability to access the database of choice and run SQL against it, and to access to the JVM of choice. The `JAVA_HOME` environment variable must also be set.

Important For a successful installation, there are several preliminary tasks that you must perform in advance. See [Before You Begin](#) for a description of these tasks.

If desired, the MoneyScape server can automatically start and stop when the computer is booted up or shutdown. MoneyScape transmitters start or stop depending on how they are configured during installation. The installation process creates the tables needed for operation in the selected database, as well as a new database user.

The following list of topics shows the sequence of steps you must perform in order to install a complete MoneyScape server. These topics are discussed in order and provide detailed instructions on each step.

Topics in this section

- [Getting Started](#)
- [Installing the Server](#)
- [Installing a Transmitter License](#)
- [Installing a Digital Certificate](#)
- [Starting the Server and Transmitter](#)
- [Installing the Developer Workstation](#)
- [Setting the Default Transmitter](#)
- [Publishing the System and Sample Channels](#)
- [Verifying the Installation](#)
- [Installing the Sample Client](#)

Getting Started

To get started, insert the MoneyScape CD-ROM into your Solaris computer and add the MoneyScape package as explained below. As explained on the following pages, the server must be installed on a Solaris machine; the Developer Workstation and the Sample client channels can be installed on Windows 95/98 or Windows NT, or on Solaris.

During the installation you can specify whether or not to restrict publishing to specific hosts. If so, the hostnames must either be resolvable using nslookup or you can add them later using the Marimba utility.

Important Keep in mind that the server installs and runs only in Solaris and the client installs and runs only in Windows. You should follow the steps in the [Installation Checklist](#) while you are doing the tasks in this section.

Configuring Transmitter Security

Transmitter security is configured during installation. A MoneyScape server can support multiple secure and non-secure transmitters. (A transmitter configured for SSL provides secure communication between client and server; a transmitter configured without SSL does not.) Both secure and non-secure transmitters can reside on the same server.

Since a MoneyScape client can subscribe to channels from more than one transmitter, a client can receive a mix of secure and non-secure channels from different transmitters. In this scenario, secure and non-secure channels are displayed in the same client shell.

Important The server installation will fail if you do not tune Solaris for an Oracle database before you begin. See [Configuring for Oracle7](#) for specific instructions.

Choosing Installation Directories

During the installation process, you must specify two Solaris directories (not to be confused with client file directories) for system files; they can use the same root path. These are the MoneyScape home directory and the MoneyScape transmitter directory. (See [What Gets Installed](#) for a complete list of the files.) *The MoneyScape home directory is always the user's home directory.*

MoneyScape Home Directory	<code>/export/home/<os_user_name></code> is usually the home directory of the transmitter user that will be the owner and runner of the MoneyScape server. There are several files and directories that will be installed, used, and managed by MoneyScape. This directory should be backed up periodically.
MoneyScape Transmitter Directory	<code>/opt/MNYSCAPE/<os_user_name></code> is the location where all files related to the transmitter will reside. These files do not change and generally require no maintenance or management.

Installing the Server

The following installation dialog shows a sample installation of a server (and transmitter) on a Sun Microsystems workstation using the Unix `pkgadd` command. Each server is associated with one transmitter. You must be logged in as `root` and you need to change to the directory where the package is located. You also need an operating system user account configured with the JDK and Oracle environments. A Sun workstation configured with Solaris 2.6, the latest Sun patch cluster, and JDK 1.1.6 or higher is required.

Important You need to create a separate operating system user account on the server for each transmitter that you install. This is the transmitter user.

Oracle7 Version 7.3.3 is installed as part of the MoneyScape server installation. If you are installing MoneyScape on a system *without* a previously configured Oracle database, the MoneyScape `pkgadd` sets all required environment variables for Oracle and the JDK. The only thing you need to do in advance is set up an appropriate Unix account name.

If you are installing MoneyScape on a system *with* a previously configured Oracle server, the server software must be Oracle7 and you will need to set the following environment variables for the transmitter user (the operating system user account dedicated to the transmitter). These variables are usually defined in the user's `$HOME/.profile` file. See [Creating a Unix User Account](#) for more details.

```
ORACLE_BASE
ORACLE_HOME
ORACLE_SID
JAVA_HOME
```

▼ To install the server

1. Carefully review the *Installation Checklist* and refer to it as you install the server.
2. If you have not already done so, become the root user and use the operating system administration utility to create an operating system user account dedicated to the transmitter. This is the transmitter user. **If Oracle is not installed on your system, skip to Step 5.**
3. Login to the system as the transmitter user configured in Step 2.


```
# su - <os_user_name> (where <os_user_name> is the operating system user name)
```
4. Verify that the environment variables describe above are configured properly for this user with the command `echo $variable_name`. If any of the above environment variables are null, modify the user's `$HOME/.profile` to contain the proper values for these variables, e.g. `JAVA_HOME=/usr/java`.
5. Become the root user.


```
$ su root
```
6. Insert the CD-ROM. It should automount to `cdrom/moneyscape`.
7. Type `cd /cdrom/moneyscape/server`
8. Type `pkgadd -d .` (Be sure to enter space and period.)

Installation Dialog

The following installation dialog is displayed on the screen. *All items in the dialog require a response; the items within square brackets generally show the default (if any).* Press Return each time you enter data, or press Return without an entry to accept the default. Note that the `<os_user_name>` shown in the dialog refers to the Unix user account name for the server.

Important The installation program prompts you for a `proxy host IP address` and verifies that you have access. If you enter a legal address for which you do not have access, the installation will hang and you must press CTRL+C to restart.

The following packages are available:

```
1  MNYSCAPE      MoneyScape Server
      (sun4) 1.0
```

Select package(s) you wish to process (or 'all' to process all packages). (default: all) [?,??,q]:

Processing package instance <MNYSCAPE> from </cdrom/moneyscape/server>

MoneyScape Server
(sun4) 1.0

Copyright 1999, MECA Software, L.L.C., All rights reserved.

This software is the confidential and proprietary information of MECA Software, L.L.C. ("Confidential Information"). You shall not disclose such Confidential Information and shall use it only in accordance with the terms of the license agreement you entered into with MECA.

Copyright 1996-1997 Marimba, Inc., All rights reserved.
Contains Castanet (tm) technology from Marimba, Inc.

Do you agree to the terms of this agreement? [y,n,?]

Enter the previously created Unix user account name that will be the owner of the MoneyScape server.

Enter a valid Unix user account name for the server:

Specify the MoneyScape home directory path of the Unix user account name above. This will be used as the directory that stores channels. There must be adequate free disk space to store the data for all channels.

Enter the MoneyScape home directory [/export/home/<os_user_name>]:

Specify the MoneyScape transmitter directory. There must be at least 20 MB of free disk space.

Enter the MoneyScape transmitter directory [/opt/MNYSCAPE/<os_user_name>]:

Specify the port number that will be used for publishing and subscribing.

Enter the publishing port number [6322]:

Specify a password that will be used for publishing.

Enter the publishing password or <ENTER> for no password:

Enter the hostnames from which you can publish to this transmitter. If no hosts are specified, any host can publish channels to this transmitter. To restrict access to this host only, specify this hostname. The hostnames must either be resolvable using nslookup or you can add them later using the Marimba utility.

Enter the hostname(s) for publishing or <ENTER> for all hosts:

Will this transmitter be configured securely with SSL-enabled? If so it will need the certificate password each time it runs.

Will this transmitter be SSL-enabled? [y,n,?]

Enter a firewall proxy host IP address for URL adapters that need an external http connection.

Enter a valid firewall proxy host IP address or <ENTER> for no firewall proxy:

Enter the firewall proxy port number for this host IP address.

Enter the firewall proxy port number [8080]:

Specify the authentication server hostname for MoneyScape administration.

Enter the authentication server hostname (e.g. HOSTNAME.DOMAIN) [server.domain_name]:

Enter the authentication server port number for MoneyScape administration.

Enter the authentication server port number:

Enter the Administrator Name for the Server Administrator tool.

Enter the administrator name [<os_user_name>]:

Enter the Administrator Password for the Server Administrator tool.

Enter the administrator password:

Specify whether you want to start the transmitter automatically at system bootup. This prompt is not shown if the transmitter was SSL-enabled above.

Start the transmitter automatically at system bootup? [y,n,?]

You have completed the MoneyScape Server setup.

Using </home_dir> as the package base directory.

Processing package information.

Processing system information.

Verifying disk space requirements.

Checking for conflicts with packages already installed.

Checking for setuid/setgid programs.

This package contains scripts which will be executed with super-user permission during the process of installing this package.

Do you want to continue with the installation of <MNYSCAPE> [y,n,?]

Messages will now scroll across the screen as the package is being installed. You will see the following message when the package installation completes successfully:

Installation of <MNYSCAPE> was successful.

This completes the `pkgadd` and the installation will run to completion. If you are installing MoneyScape and Oracle7 Server from a 2X CD-ROM, this may take 30–60 minutes.

What Gets Installed

The files added to your system with the `pkgadd` program are listed and explained below. As noted, these files are stored in the MoneyScape home directory [`/export/home/<os_user_name>`] and in the MoneyScape transmitter directory [`/opt/MNYSCAPE/<os_user_name>`] respectively. Note that many of these files are not actually visible until you run a transmitter.

MoneyScape Home Directory [<code>/export/home/<os_user_name></code>]	
<code>channels, compress, files, diffs</code>	Used to store published files and should not be changed or modified directly. They are managed by MoneyScape.
<code>bin</code>	This directory contains various shell scripts and SQL scripts used by MoneyScape. Files starting with <code>ORA_</code> and ending with the <code>.sql</code> extension are SQL scripts that are used to create and populate the tables used by MoneyScape. <code>ORA_All.sh</code> and <code>ORA_Reference.sh</code> are scripts that run all the SQL scripts in the proper order during install time. These could be used to rebuild the database if you need to rebuild from scratch. <code>admin.sh</code> is a script used to run the Server Administrator. <code>runRmiServer.sh</code> is called by an <code>/etc/init.d</code> script to run the server; <code>stopRmiServer.sh</code> is called to stop the server. <code>transmitter</code> is a script that is used to start/stop the transmitter. It is called by the Server Administrator and should not normally be called directly. <i>If called directly, the user calling it must be the user that this transmitter was installed under.</i>
<code>etc</code>	This directory contains property files and any other miscellaneous files used by MoneyScape. These files are generated during install time. <code>AdminProperties.txt</code> contains information used by the Server Administrator. <code>DataBase.txt</code> contains database connect information and passwords. <code>PluginProperties.txt</code> contains proxy information. <code>PoolProperties.txt</code> contains entries which dictate the size of various object pools that can be adjusted for optimum performance. This should be done only in consultation with MECA. <code>UserAuthentication.txt</code> contains information about who is allowed to perform what administrative tasks.
<code>lib</code>	This directory contains the JAR files needed to run MoneyScape. Also included are JDBC drivers and a shared library for JDBC to connect to Oracle.

MoneyScape Home Directory [/export/home/<os_user_name>]	
<code>logs</code>	This directory contains various log files that can be used for error detection. <code>MecaServer.out</code> logs any messages or output from the server. This file should be examined if there are problems running the server. <code>errorlog</code> logs errors and exceptions thrown by the transmitter. This file can be viewed from the Server Administrator. <code>java.out</code> contains messages, information and errors from the server and can also be viewed from the Server Administrator. <code>log</code> contains information about who connected to a transmitter, and <code>publish.log</code> contains all published information.
<code>outside_classes</code>	This directory is the location in which you should place any classes you write as additions to the MoneyScape server, most notably adapters. For example, if you write an adapter called <code>com.bankname.MyAdapter.java</code> , it should be placed in <code>outside_classes/com/bankname/</code> . It will be then accessible to MoneyScape when configured with the Server Administrator.
<code>Other files</code>	A boot time start and stop script is created in <code>/etc/rc3.d</code> . These are linked to <code>/etc/init.d/mecaserver_user</code> , where <code>user</code> is the transmitter user that was created for MoneyScape. To start or stop the MoneyScape server as <code>root</code> : <ul style="list-style-type: none"> • <code>type /etc/init.d/mecaserver_user start</code> to start the server. • <code>type /etc/init.d/mecaserver_user stop</code> to stop the server. There is a file created called <code>/etc/mecaserver_user.conf</code> which contains information used by <code>/etc/init.d/mecaserver_user</code> to start.
<code>license.dat</code>	This is the file used by the transmitter to determine the number of clients allowed to connect based on your MoneyScape license agreement. This can be replaced with a new license if needed. If not present, a temporary, five-user license is automatically created.
<code>properties.txt</code>	This file is used by the transmitter to determine startup information like, for example, the port number for the connection. This should not normally need any intervention by an administrator.
<code>users.dat</code>	This file is maintained by the transmitter and used to determine information about connected users (the number at any given time) and is used for licensing purposes.

MoneyScape Transmitter Directory [/opt/MNYSCAPE/os_user_name]	
<code>Marimba Castanet bundle</code>	The Sun JDK and all files used by the Marimba transmitter.

Uninstalling the Server

If you want to remove the server software, or if you want to upgrade to a newer version, you must first run `pkgrm` to remove the previous package. Be aware that `pkgrm` does not remove all installation files; some residual files must still be removed manually. *Note also that if you are running multiple MoneyScape servers, a `pkgrm` will remove them all.*

▼ To remove a package

Type `pkgrm MNYSCAPE`

The following package is currently installed:

```
MNYSCAPE      MoneyScape Server
                (sun4) 1.0, RC2
```

Do you want to remove this package?

Backing Up the Server

You should be performing a full system backup (including all MoneyScape files) according to the backup schedule in place at your site. A backup of these file systems includes the Oracle database that is installed with MoneyScape. For specific recommendations and detailed procedures about backing up the Oracle database, see the Oracle documentation.

Installing a Transmitter License

You must install a transmitter license, supplied by MECA, for each transmitter. Before server installation, contact your account representative at MECA for a license. MECA will e-mail you a `license.dat` file within 48 hours that you copy to the `home` directory of the transmitter user. This file is read when the server starts.

▼ To install a transmitter license

1. Backup the `license.dat` file for safekeeping in case you need to re-install.
2. Copy the `.dat` file to the `home` directory of the transmitter user.

Installing a Digital Certificate

Important Skip this section and go to [Starting the Server and Transmitter](#) if you did not configure the server for SSL during installation. Digital certificates are only required for secure transmitters.

Digital certificates provide security protection for the client/server connection. *Currently, VeriSign, Inc. is the only certificate authority used by Marimba products.* A digital certificate remains valid for a limited time and takes 3–5 days to obtain. As a convenience, Marimba and VeriSign have created a test certificate that provides channel signing and SSL certification in one certificate. A test certificate is free and it expires two weeks after it is issued. See the FAQs for more information. Contact VeriSign for more information.

▼ To install a digital certificate

1. Get a certificate from VeriSign. (You must have a browser installed and available before you can request a new certificate.)
2. Type the following to run the Castanet Transmitter Administration utility from the MoneyScape transmitter directory. Run as the transmitter user; do not run as root.

```
opt/MNYSCAPE/transname/castanet/transmitter/bin/transmitter
```

3. Follow the instructions to cut and paste the certificate data into the Marimba Transmitter panel. As a precaution, you may want to export and save the certificate for later use.

Important Remember the password that you are asked for when installing the digital certificate. You will need this same password to start a transmitter configured for SSL.

Importing a Certificate

If the host being used to install the MoneyScape server already has a SSL Certificate for a different application (e.g. Netscape), there is usually a facility in the application that lets you export the certificate. After export, you can import it into MoneyScape as follows.

▼ To import an existing certificate

1. Run the Castanet Transmitter Administration utility. This can be accessed from the MoneyScape transmitter directory.
2. From the SSL Certificates panel, click Import Certificate.
3. Enter the name and path of the exported certificate (to import it into the MoneyScape transmitter).
4. When done click Stop and Exit.

Starting the Server and Transmitter

At this point you need to start the server and the transmitter in order to continue with the installation of other MoneyScape components.

▼ To start the server and the transmitter

1. Become the transmitter user.
2. Type `bin/runRmiServer.sh` to start the server.
3. Type one of the following commands to start the transmitter depending on whether or not it was configured for SSL:
 - `bin/transmitter startsecure <password>` (SSL)
 - `bin/transmitter start` (non-SSL)

To verify that the server has started, type the following command and check that the server processes are running.

```
ps -fu <os_user_name>
```

After the MoneyScape tools are installed, you can start a transmitter from Solaris or from Windows using the Server Administrator tool.

- From the Solaris, change to the directory where the tools are installed and type `adminTool.sh`.
- From Windows, navigate to the Server Administrator from the Start menu.

Important During server installation, the Server Administrator is installed by default in `/export/home/<os_user_name>/bin`. If you are logged in as the transmitter user, type `bin/admin.sh` to run the Server Administrator tool from the Solaris server.

Stopping the Server and Transmitter

▼ To stop the server

1. Become the transmitter user.
2. Type `bin/stopRmiServer.sh` to stop the server.
3. Type `bin/transmitter stop` to stop the transmitter.

Important If you are root, a start script and stop script are created in `/etc/rc3.d`. These are linked to `/etc/init.d/mecaserver_user`, where `user` is the name of the user that was used to install MoneyScape. Use the `start` or `stop` option after the command.

Installing the Developer Workstation

The Developer Workstation includes the tools, documentation, sample channels, and other components that are used by channel developers. As explained below, you can install these components on Windows or Solaris machines. The Developer Workstation includes:

- Tools – Alert Editor, Trigger Editor, Publisher, and Server Administrator.
- Documentation – All MoneyScape documentation in HTML and/or PDF format including javadocs (HTML only).
- Client Files – Sample client files that you can customize with your own look and feel. For example, you can customize the client so that it runs with your own logo.
- Channels – System and Sample channels. These channels must be installed and published to the transmitter on the server.

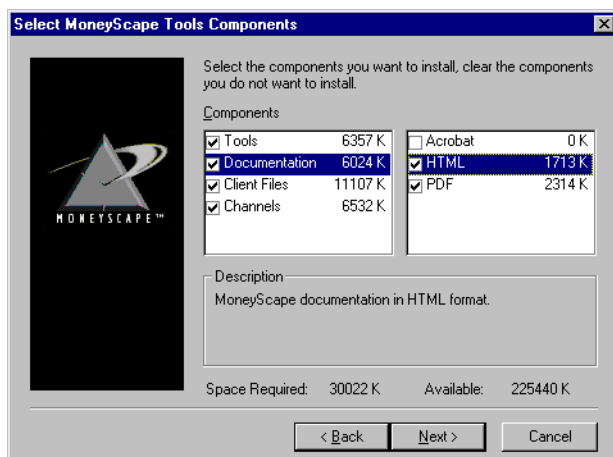
A Typical installation includes all of the above (except the Adobe Acrobat files or the MoneyScape PDF documents). A Custom installation includes only the items you select. The installation dialog varies depending on whether you are running in Windows or Solaris. When you install the Developer Workstation in Windows, you are prompted for an installation directory, and the type of installation: Custom or Typical. If you select Custom, you can select specific MoneyScape components. In Solaris, only those components that you specify are installed.

Important If you want MoneyScape documentation in PDF format (and/or an Adobe Acrobat Reader), you must perform a Custom installation and select these items. The Acrobat installation starts as soon as the Developer Workstation finishes installing.

Windows Installation

▼ To install the Developer Workstation in Windows

1. Go to the `install` directory and double-click `setup.exe`.
2. Select Custom or Typical. If Typical, all components (except as noted above) are installed. If Custom, only the components you specify are installed.



Solaris Installation

In Solaris, you are prompted for an installation directory, and must select individual components; there is no typical installation. Use standard Unix techniques to select components: use the arrow keys to navigate, the spacebar to select, and the Enter key to drill down. *Note that you must be logged in as root to install Adobe Acrobat in Solaris.*

▼ To install the Developer Workstation in Solaris

1. Open an Xterm window.
2. Type `cd /cdrom/moneyscape/install` to change to the directory where the tools are installed.
3. Type `install.sh` to install the selected components into the specified directory.
4. Select the specific components that you want to install as explained on the following pages.



Installing the Tools

You generally install the tools on the system from which MoneyScape channels will be published. The tools are briefly explained below. The Server Administrator is described in detail in [Using the Server Administrator](#); the other tools are explained in the *Channel Developer Guide*.

Tool	Description
Server Administrator	manages the server environment and broadcasts channels to end users. The <code>pkgadd</code> also installs this program on the server.
Publisher	used by channel developers to manage the publication of channels. It sets properties and publishes channels to a transmitter for broadcast to end users.
Trigger Editor	used by channel developers to fire events that are based on actions that occur within a channel and can be set in advance. For example, clicking an icon can display a specific HTML page.
Alert Editor	used by channel developers to add messages to the “Important” menu or to the ticker.

Launching the Tools

In Windows, you can launch the MoneyScape tools from the Start menu. In Solaris, change to the directory where the tools are installed and type the appropriate script.

- Alert Editor – `alertTool.sh`
- Publisher – `publish.sh`
- Server Administrator – `adminTool.sh`
- Trigger Editor – `triggerTool.sh`

Installing the System and Sample Channels

The System and Sample channels must be installed and published to a running transmitter as part of the MoneyScape installation. They are both installed during a Typical installation. This section only explains how to install the channels; to publish the channels, see [Publishing the System and Sample Channels](#).

Important Both the System and Sample channels must be installed and published as part of the MoneyScape installation. If you are simply installing the Developer Workstation or the Sample Client, you don’t have to install or publish these channels.

You can install the system and sample channels in Windows or in Solaris depending on where you will be publishing them from. *Note that the System and Sample channels must be in a writable directory on disk; you can’t publish these channels from the installation CD-ROM.*

Installing the Documentation

You can install MoneyScape documentation on a Solaris server or in Windows. The documentation is available in HTML and Adobe PDF format. The HTML documents run in a browser; the PDF files require an Adobe Acrobat Reader. If you need to install an Acrobat Reader, be sure to check the appropriate box and the Reader installation will launch automatically.

For easier hard-copy reading, you can print the PDF files from the CD-ROM or from wherever you install the files. *For best results, be sure you print to a device that is configured for PostScript printing.*

ChanDevGuide	Explains all aspects of channel development, including how to write and publish Java channels and HTML channels, how to write adapters that interface with external data sources, and how to use the MoneyScape tools.
ConceptsGuide	Describes the conceptual framework behind MoneyScape and explains how to successfully exploit key features.
ClientCustomizationGuide	Explains how to brand and customize the MoneyScape client application.
InstallChecklist	Provides a quick overview or snapshot of the installation process. Also describes the contents of the CD-ROM and the Documentation folder.
ServerAdminGuide	This administrator guide.
javadoc	The javadocs for all extensible MoneyScape Java packages. This is the primary interface to the MoneyScape API. To view the javadocs, open packages.html in your browser.

Important If don't install the MoneyScape documentation, you can open a browser and view the HTML documentation and javadocs from the installation CD-ROM. If you copy any documentation to your system, copy the entire directory structure to preserve the links.

Installing the Client Files (Windows only)

The client application only runs in Windows and the files can only be selected and installed in Windows. The sample client files are used to customize the look and feel of the client. For example, if you change the background image in the `properties.txt` file, your own logo can be displayed when you subsequently run the client. For a complete description of how to customize the client shell, see the *Client Customization Guide*.

The sample client files are installed in `homedir/lib` and include the following.

- `palette.txt` – Set fonts and colors
- `lookfeel.txt` – Assigns fonts and colors to specific elements
- `properties.txt` – Sets the background image, client layout, and other properties.

Setting the Default Transmitter

Before you publish the system and sample channels, you must set the default transmitter on the server to the machine name (*not the transmitter name*) and port number or the IP address and port number of the transmitter you just installed.

▼ To set the default transmitter

1. Go to `homedir/channels/SystemChannels/MoneyScapeClient/properties.txt` and edit the following line: `defaulttrans0=your.transmitter.spec.goes:here`

2. Replace `your.transmitter.spec.goes:here` with the machine name where you installed the transmitter. The machine name can be entered in one of the following formats. Note that the port number must correspond to the publisher port that you specified during server installation.
 - the IP address and port number, e.g. `192.168.2.20:2020`
 - the fully qualified hostname (hostname and domain name) and port number, e.g. `fasholt.mymnet.com:2020`

Important If you configured the transmitter for SSL during installation, you must use the fully qualified hostname; an IP address is invalid.

Publishing the System and Sample Channels

The System and Sample channels must be installed and published as part of the MoneyScape installation. (You don't have to install or publish these channels if you are simply installing the Developer Workstation or the Sample Client.) They must be installed in a writable directory on disk because you can't publish these channels directly from the CD-ROM.

When you run the Publisher, the window is automatically populated with the system and sample channels that you installed. Publish each channel separately as explained in the Channel Developer Guide; don't change any fields.

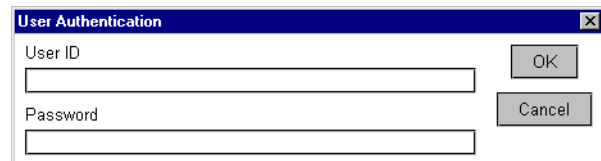
You can start the Publisher from Solaris or from Windows. From Solaris, change to the directory where the tools are installed and type `publish.sh`. From Windows, navigate to the Publisher from the Start menu.

Important Once installed, you must publish these channels with the Publisher. The system channels and the sample StartupChannel are required. The remaining Sample channels are for demonstration purposes and are optional.

Selecting a Startup Channel

Before you can run the MoneyScape client, you need to install and publish a startup channel. The client won't run otherwise. If you installed the Sample channels you can publish the `StartupChannel`, the `EnrollmentStub`, or both. They are both startup channels.

The `StartupChannel` displays a typical client enrollment dialog that prompts for a User ID and Password. When the client initially runs, it defaults to this channel. This demonstrates a typical client enrollment procedure that utilizes

A screenshot of a 'User Authentication' dialog box. It has a title bar with the text 'User Authentication' and a close button (X). The dialog contains two text input fields: 'User ID' and 'Password'. To the right of the 'User ID' field is an 'OK' button, and to the right of the 'Password' field is a 'Cancel' button.

MoneyScape security features and can be customized as appropriate. The acceptable values for User Authentication are `userid1-9` and `password1-9`.

Also included in the Sample Channels is the `EnrollmentStub` channel. This startup channel is used for development purposes only and bypasses the userid and password dialog entirely. **For security purposes, never publish the EnrollmentStub sample channel to a production**

transmitter. If you want to use the EnrollmentStub, set the `enrollchan=EnrollmentStub` in `MoneyScape/lib/properties.txt`.

Verifying the Installation

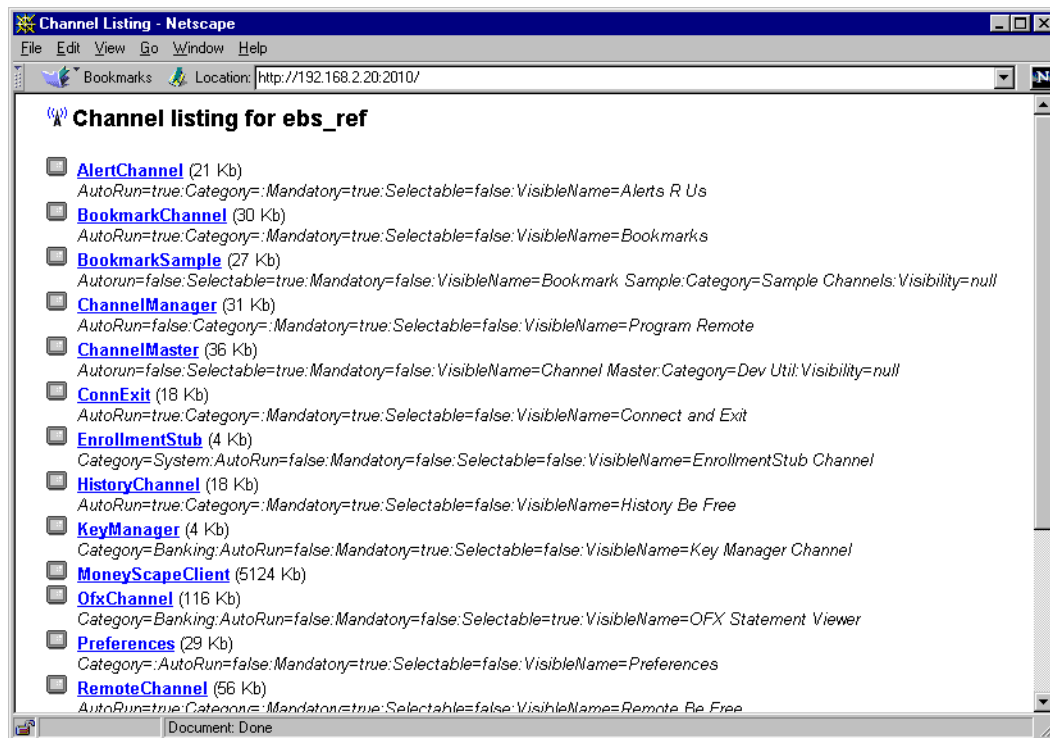
To verify a successful installation, open your favorite browser (in Windows or Solaris) and type in the server IP address followed by a colon and the transmitter port number. For example,

`http://192.168.3.20:3030` for a non-secure transmitter.

`https://192.168.3.20:4040` for a secure transmitter.

If a window similar to the one shown below is displayed, it means that the transmitter is running and the channels have been published. The server installation has been successful and you are done. In Solaris, you can also type the following command to verify that the installed channels are running.

```
ls $HOME/channels
```



Installing the Sample Client

The sample client application only runs in Windows. By default, the sample client is installed in `C:/Program Files/MoneyScape`.

▼ To install the MoneyScape client

1. Insert the CD-ROM on a Windows machine.
2. Go to `client/win32`, double-click on `setup.exe` and follow the prompts.

3. Set the default transmitter as explained below.

Setting the Default Transmitter

- ▼ To select a default transmitter.

1. On the Windows machine where MoneyScape is installed, open the file `moneyscape/lib/properties.txt` and find the following line.
`defaultttrans0=your.transmitter.spec.goes:here`
2. Edit this line to specify the transmitter on your MoneyScape server. The format differs depending on whether or not the transmitter was configured for SSL:
 - SSL – specify host name and port number, e.g:
`defaultttrans0=yourhostname.yournetwork.com:1010`
 - non-SSL – specify host name and port number or host IP address and port number e.g:
`defaultttrans0=123.456.7.89:1010`

Running the Sample Client

Once you have set the default transmitter, you only need to select “MoneyScape” from the Start Menu Programs to run the sample client. When prompted, click Go Online, and then enter a User ID and Password; the acceptable values are userid1–9 and password1–9. When the application starts, click on the Help channel and choose the MoneyScape Tutorial. This explains how to set preferences, subscribe to channels, etc.

Using the Server Administrator

The MoneyScape Server Administrator is an application you use to manage the MoneyScape server. The Server Administrator configures and manages the relationship between adapters, datafeeds, and channels running in the MoneyScape server application. The Server Administrator lets you make dynamic configuration changes to transmitters during runtime, without starting or stopping the server.

The Server Administrator enforces user authentication by verifying the login name and password. It starts and stops transmitters, and it builds metadata records for channels and datafeeds. MoneyScape may consist of multiple servers that are each associated with one transmitter. The Server Administrator only manages the server that you are presently connected to via the login window. Note that you can't run the Server Administrator from behind a firewall.

This section provides an overview of server concepts and capabilities. For an explicit discussion of how to configure channel and datafeed records, see [Configuring Channels and Datafeeds](#).

Topics in this section

- [How the Server Administrator Works](#)
- [Logging In](#)
- [Using the Server Administrator](#)

How the Server Administrator Works

The Server Administrator consists of two pieces of client/server software that work together. One piece resides on the client machine where the Server Administrator is running; the other resides on the server machine where the transmitters are located. This can actually be the same machine or different machines. The Server Administrator client communicates with a running server via the Java Remote Method Invocation (RMI). There is authentication for administrator permissions, and any command is authenticated by the designated server before execution.

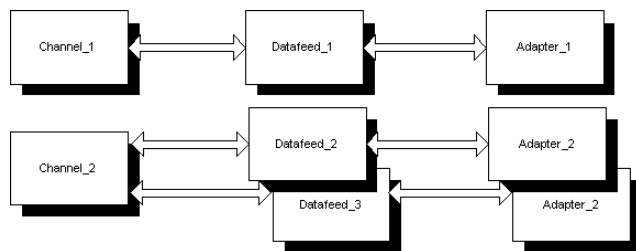
Important When using the Server Administrator to configure datafeeds and channels, the data transmitted from the Server Administrator client to the server is not encrypted and neither are SSL certificate passwords.

Adapters, Datafeeds, and Channels

MoneyScope communicates with outside data sources using *adapters*. Adapters are the hookpoints into the server from outside data sources such as legacy mainframes, data warehouses, etc. This external data is then available to channels. Adapters are designed to be extended into specific protocols and can be customized to get data from the actual source. MoneyScope provides pre-built adapters, and you can develop your own adapters. See the *Channel Developer Guide* for more information.

A *datafeed* manages data that is on the server. It caches, fetches, and distributes data. It is the standard interface through which requests for information are routed to and from external adapters. Datafeeds work with channels and adapters within MoneyScope. They manage the data that has been returned from an adapter and feed it to channels (which are Java or HTML applications that run in the MoneyScope client shell).

Multiple datafeeds can be associated with one channel, but there is a 1:1 relationship between adapters and datafeeds as shown in the following diagram. Similarly, there may be multiple instances of the same adapter, but each is associated with only one datafeed.



Channels use the data from a datafeed. There can be multiple datafeeds feeding into one channel. To add channels, you use a Server Administrator menu item to add a new channel (or datafeed) to the server. **The channel name on the server must exactly match the channel name as defined in the MoneyScope Publisher.** This is the only association between a channel published with the Publisher and a channel record residing on the server.

Adding a channel may also require associating that channel with an adapter. For example, a stock quote channel may send stock symbols chosen by the client to the server so that data can be retrieved through an external adapter. The channel displaying the stock quotes must be associated with the adapter (i.e. the datafeed) supplying the quotes.

Authentication Server

In MoneyScope, there is one authentication server, defined during installation, that is designated to manage authentication, permissions, etc. for the entire system. As noted, the Server Administrator uses the Java Remote Method Invocation (RMI) as the communications protocol to communicate with the server. RMI lets you remotely retrieve and run Java objects stored on a network.

Each MoneyScope server runs an `rmiregistry` to which clients have access. The server makes this object available to clients. This registry acts as a directory service for remote objects that are available to the server. The registry keeps track of datafeeds and channels existing on the server and notifies the Server Administrator. MoneyScope clients connect to the registry through the specified RMI port.

Admin Properties

As noted, the Server Administrator consists of two pieces of client/server software. One piece resides on the client machine where the Server Administrator is running; the other resides on the server where the transmitters are located. Both have an `AdminProperties.txt` file. On the server, this file is in `homedir/etc`; on the client, this file is in `MoneyScape/tools/etc`.

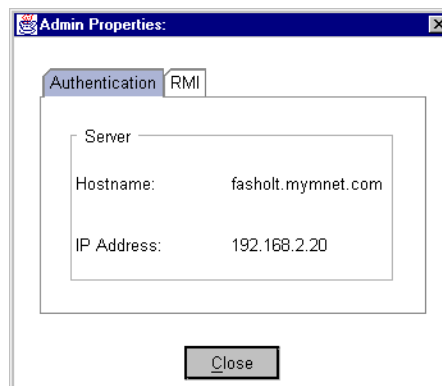
The `AdminProperties.txt` files are automatically generated during installation and have settings that define specific configuration parameters that are used by the Server Administrator. If necessary, you can use a text editor to modify these files. The `AdminProperties.txt` files reside in the `etc` directory on both the client and the server respectively. (The server file has two additional settings as explained below.)

<code>rmiRegPort</code>	Client and server. The server creates an RMI Registry to run on the specified port. Defaults to 1099. You can override this value from the Login window.
<code>authHostName</code>	Client and server. Specifies a fully qualified hostname designated to manage user authentication. You can override from Login window.
<code>admin.logs.maxbytes</code>	Client only. The maximum number of bytes (in <code>errorlog</code> and <code>java.out</code>) that are transmitted back to the client and displayed in the selected log window. The default is 8 KB. You can add this setting with an editor if not present.
<code>startSecureTransmitter</code>	Server only. If true, the transmitter is SSL-Enabled.
<code>autoStartTransmitter</code>	Server only. If true, a non-secure transmitter autostarts when the server reboots.

The `AdminProperties.txt` file is automatically generated during server installation and provides information needed to run the Server Administrator. The Admin Properties window shows the settings in this file. In MoneyScape, there may be multiple servers on the same local network. *Be sure that you are logged into the right server on the network.*

▼ To display Admin Properties

1. Select Properties from the Admin submenu to display the IP address and RMI port of the specified host server.

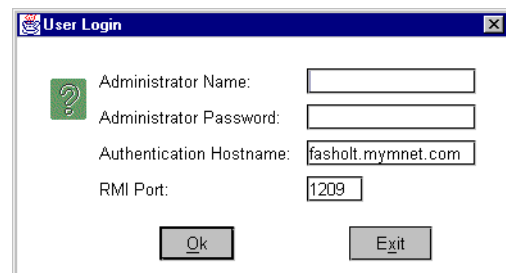


Logging In

You can start the Server Administrator from Solaris or from Windows. From Solaris, change to the directory where the tools are installed and type `adminTool.sh`. From Windows, navigate to the Server Administrator from the Start menu.

Important During server installation, the Server Administrator is installed by default in `/export/home/<os_user_name>/bin`. If you are logged in as the transmitter user, type `bin/admin.sh` to run the Server Administrator tool from the Solaris server.

The login window is the first screen displayed when you start the Server Administrator. The name and password are assigned during server installation. There is one user name per system. The password can be modified with the Change Password command on the Admin menu. After you login successfully, the Server Administrator main window is displayed.

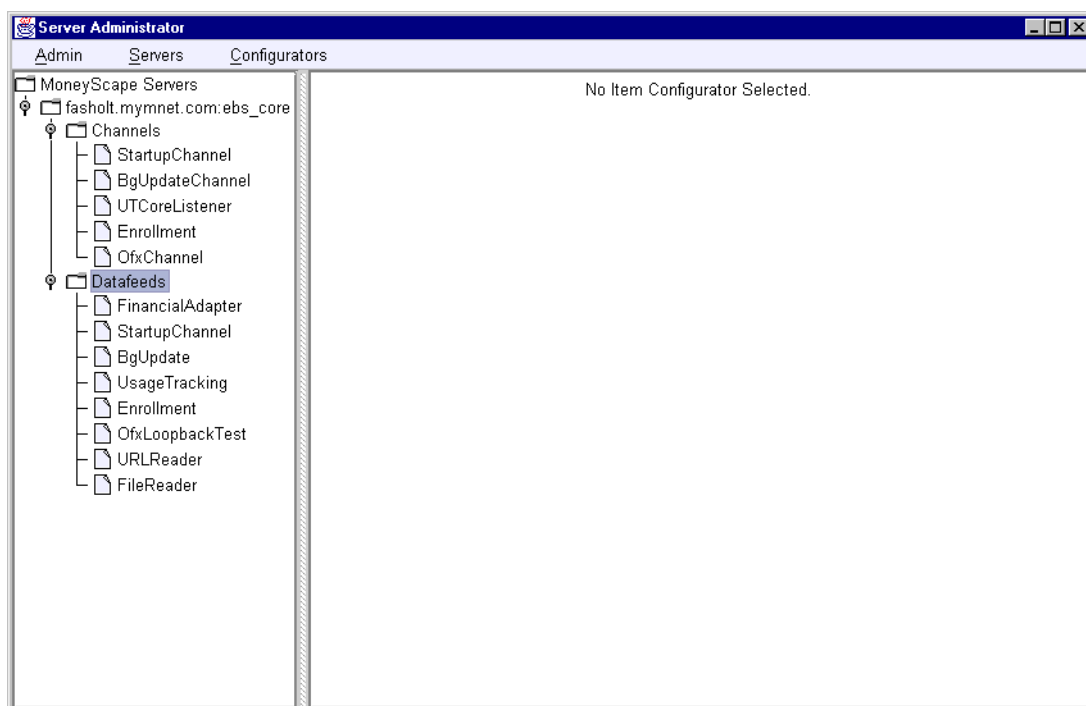


Administrator Name	There is one Server Administrator name that is assigned during server installation. To change this name, you must edit the appropriate file (contact MECA for instructions) or re-install the server software.
Administrator Password	There is one Server Administrator password that is assigned during server installation. This password can be modified with the Change Password command on the Admin submenu.
Authentication Hostname	The fully qualified hostname (hostname and domain name) of the server that was designated for user authentication during server installation. This is the server that you want to connect to. <i>Do not specify an IP Address.</i> The Server Administrator contacts this authentication hostname to verify users at login and before performing actions for which permission is required.
RMI Port	The port number where the RMI Registry will run. There is one rmiregistry for each server. Each server creates an RMI registry to run on the specified port. Defaults to 1099.

Using the Server Administrator

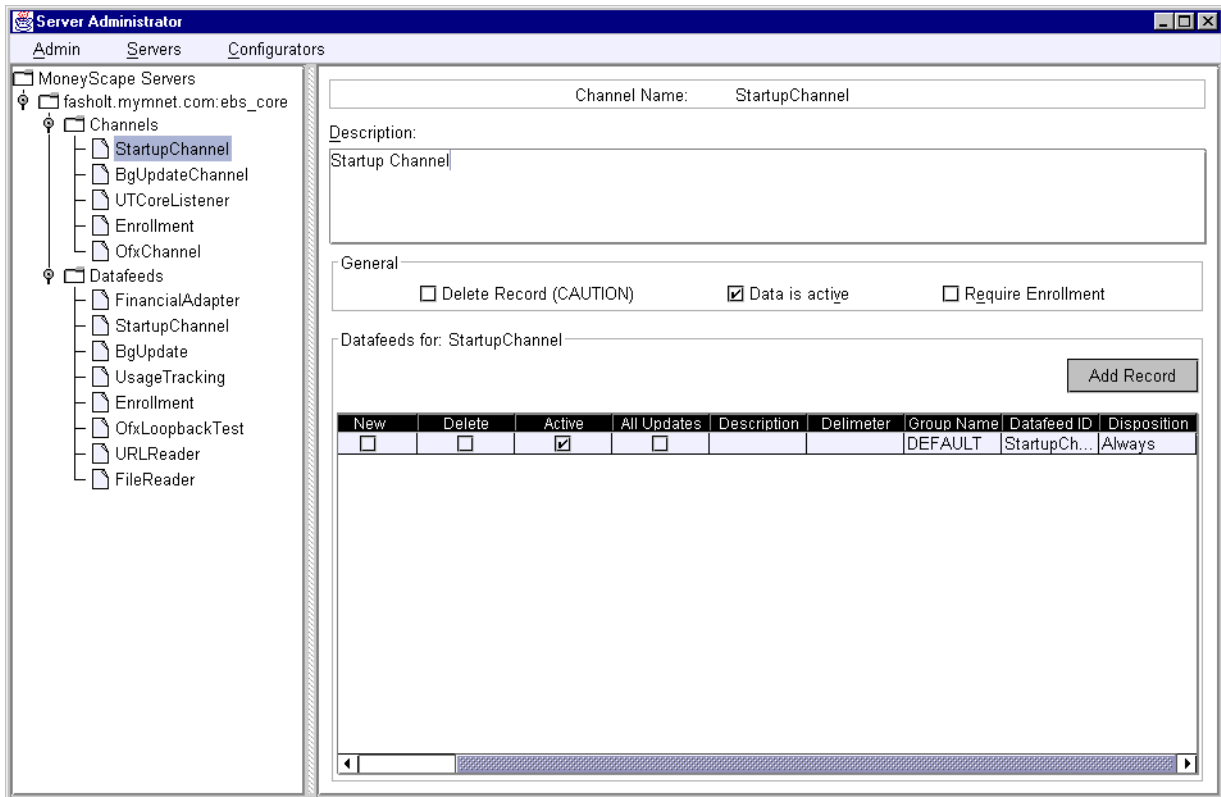
The Server Administrator main window shows the MoneyScape server applications. The tree control in the left pane shows the channels and datafeeds for the server on which you logged in.

In MoneyScape, a server application name (i.e. `fasholt.mymnet.com:ebs_test`) consists of three parts: a hostname (`fasholt`), a domain name (`mymnet.com`), and a transmitter name (`ebs_core`). The fully qualified hostname consists of the hostname and the domain name (i.e. `fasholt.mymnet.com`). A fully qualified hostname resolves internally to an IP address.



Admin	<ul style="list-style-type: none"> • Properties – Quickly shows Hostname, IP address, and RMI port number from the <code>AdminProperties.txt</code> file. • Change Password – Changes the single login password needed by anyone using the Server Administrator. • Logout – Logs out the current user and leaves the current Server Administrator session running. Displays the Login window so that a new user can login. • Exit – Exits the Server Administrator entirely.
Servers	<p>Rescan – Re-reads permissions from the server and re-validates Administrator Name and Password. Refreshes the screen as new channel or datafeed records are added.</p>
Configurators	<ul style="list-style-type: none"> • New – Use to configure a new channel or datafeed for the specified host (transmitter). • Commit – Saves changes in database to the source server after creating or modifying a configurator.

Click on any discovered channel or datafeed to display the corresponding configurator for that item. For example, the right panel below shows the configurator for the Startup Channel. The configurator windows are the same for all channels and datafeeds respectively, but the information they contain is different. See [Configuring Channels and Datafeeds](#) for a description of the configurators and what the data fields mean.



Starting and Stopping a Transmitter

To start a transmitter, you may need to enter a password depending on how the transmitter was configured during installation. If the transmitter was SSL-Enabled, the Start button appears dimmed; if the transmitter was not SSL-Enabled, the Secure Start button appears dimmed. To stop a transmitter you simply right-click on the server application name and choose Stop from the pop-up menu.

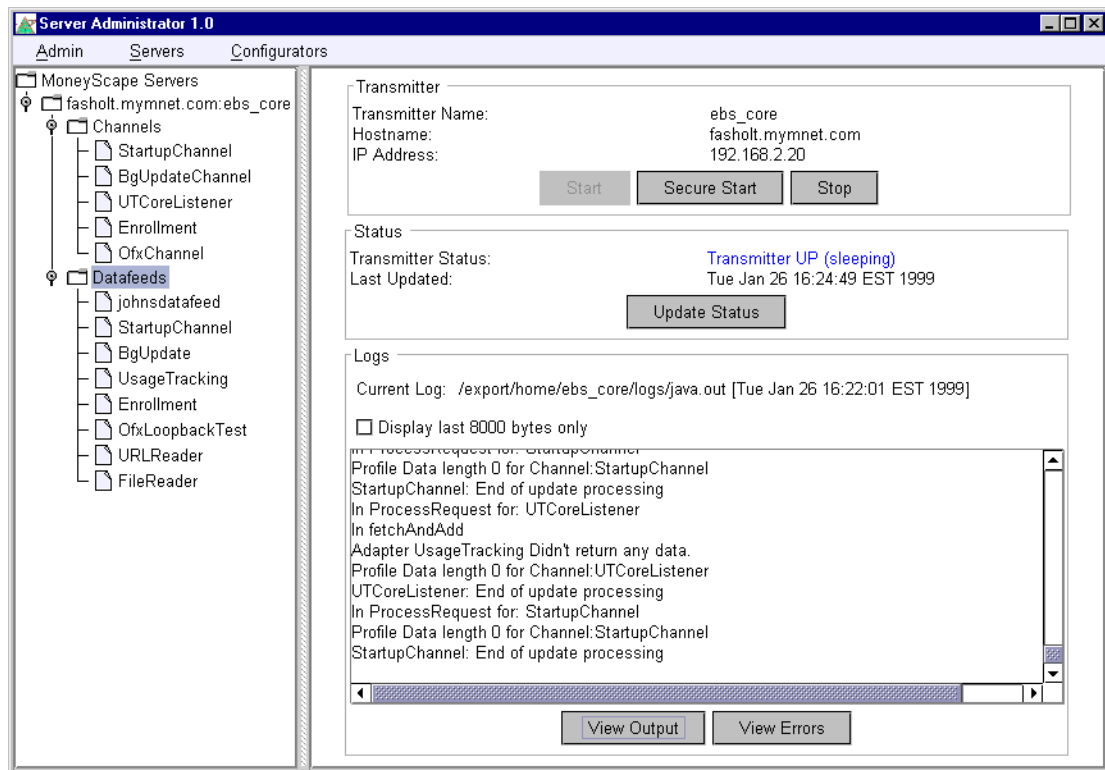
▼ To start or stop a transmitter

1. Right-click on the transmitter name (e.g. `fasholt.mymnet.com.ebs_core`)
2. Select Start/Stop Transmitter.
3. For a Secure Start (if the transmitter was configured with SSL-Enabled during server installation), you are prompted for the password that was created when you installed the digital certificate.

Autostarting a Transmitter

During installation, you have the option to configure a non SSL-Enabled transmitter for autostart after a reboot. (You can't configure an SSL-Enabled transmitter for autostart because a password is always required.) If you didn't originally choose autostart, you can change to autostart (or no autostart) after a reboot by editing the `etc/AdminProperties.txt` file on the server as follows:

```
autoStartTransmitter=true
autoStartTransmitter=false
```



Start	Starts a transmitter that was installed without SSL Enabled.
Secure Start	Starts a transmitter that was installed with SSL Enabled. Prompts for the password specified when the digital certificate was installed.
Stop	Stops the specified transmitter.
Update Status	Click to update the transmitter status.
Display last 8000 bytes only	If checked, the last 8 KB in the selected log is displayed. This value is configurable in <code>AdminProperties.txt</code> on the client.
View Output	Displays the log of server events (<code>java.out</code>). This log also displays debug data generated by channels if the channel is configured for debugging.
View Errors	Displays log of transmitter events (<code>errorlog</code>).

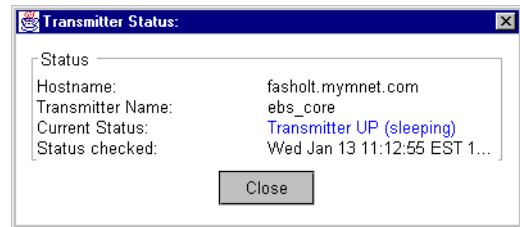
Important When attempting to View Output or View Errors, you may experience a slight delay as the system loads the entire log file. To avoid any delay, check “Display Last 8000 bytes only” to display only the most recent entries in the file.

Transmitter Status

The Transmitter Status window provides a quick snapshot. It is simply a subset of information from the previous window.

▼ [To check transmitter status](#)

1. Right-click on the transmitter name.
2. Select Status.



Configuring Channels and Datafeeds

In MoneyScape, a channel is a path to an end-user client machine and a datafeed is a path to cache or to an external data source. An adapter is an interface between the datafeed and an extension that actually retrieves the external data. This section explains how to create channel and datafeed records for channels that are built by developers.

Adapters and datafeeds work together and can be viewed as a single entity. (This is explained in more detail in [Adapters, Datafeeds, and Channels](#).) The Server Administrator is the application used to manage these channels and datafeeds. It uses channel records and datafeed records to manage these objects. For this to happen, you need to create corresponding channel records and datafeed records using the Server Administrator.

- You must create a channel record for any channel that has a datafeed, that is for any channel that accesses external data. *If a channel doesn't have a datafeed, the channel record is optional.*
- You must create a datafeed record for every datafeed.

Note that all channels can use datafeeds; they are not confined to Java channels. For example, an HTML channel can be configured to fetch new HTML pages from an external source at regular intervals. It needs a datafeed to do this.

The Configurators menu item on the Server Administrator lets you dynamically add or change channel and datafeed records and commit them to a server. A key feature of MoneyScape is that you can dynamically add or change channels and datafeeds without starting or stopping the server.

Topics in this section

- [Configuration Overview](#)
- [Creating Channel Records](#)
- [Creating Datafeed Records](#)
- [Configuration Examples](#)

Configuration Overview

All of the following basic steps are typically needed to configure and publish a channel that can access external data. Many of the steps may be done concurrently, by different developers, and the actual sequence may vary, but each step is required.

1. Define your requirements in the end user channel to which a user may subscribe.

2. Decide what the client and adapter exchange (Strings, serialized objects, arrays of objects, etc.).
3. Write the adapter code (if a pre-built adapter is not adequate) as explained in the *Channel Developer Guide*.
4. Add the adapter to the `outside_classes` directory on the server and then associate it with a datafeed using the Server Administrator.
5. Again using the Server Administrator, add a channel record for the channel and a datafeed record for the channel.
6. Write the Java code for the channel.
7. Publish and test the channel.

How Adapters Work

To create the adapter, you implement `IAdapter` as explained in the *Channel Developer Guide*; this makes the adapter usable by the datafeed. The backend of the adapter (the extension) is custom code, created by a channel developer. Then you run the Server Administrator and enter specific metadata such as the datafeed name, how often the data changes, etc.

All external adapter files must reside in the `outside_classes` directory which is created on the server when the system is installed. This directory includes pre-built adapters `URLAdapter` and `FileReaderAdapter` and is also used for additional adapters created by developers. The Java code that implements `IAdapter` must be written and added to this directory; this is where the server looks for external adapters.

There are three interrelated steps in adapter development:

1. Write a new adapter (or use a pre-built adapter) for access to external data.
2. Put the adapter file in the `outside_classes` directory.
3. Configure the channel and datafeed records.

When the MoneyScape server starts, it first reads the property files in the user's `etc` directory. These files provide information on such things as how to connect to the database, proxy information, and various startup parameters. Next, it reads the metadata (if applicable). Metadata is data about data. For transmitters, this includes information about channels, adapters, and the characteristics of data being stored on the server. This metadata is configurable and can be dynamically changed while the server is running. This means you can add, delete, and modify channels and datafeeds, and their behavior, without stopping the server.

When you commit a change, the metadata is changed, any interested parties (i.e. running transmitters) are notified, and the changes are logged. This log is located in the MoneyScape `logs` directory and named `MetaDataChanges.log`. It is written in SQL format so that it can be used to re-apply these same administrative changes at some later time.

Data That Adapters Retrieve

During channel updates, various types of data are passed from the server to the client. When configuring channel records and datafeed records as explained on the following pages, it is important to understand the different types of data that are accessible from the server during channel updates.

Adapter Data Types	Description
Generic Data	<i>Static data distributed to all clients. This is distributed to all clients that are subscribers to a given channel during an update. Examples are Java class file, gifs, HTML code, etc.</i>
Custom Channel Data	<i>Dynamic data distributed only when requested. This is dynamic data that is available to all clients that are subscribers to a given channel but it is only sent to a user when specifically requested. For example, a company news channel might contain news stories for most publicly traded companies. Rather than getting data for all publicly traded companies, an end user can receive a subset of the data (e.g. IBM and Netscape).</i>
User Specific Data	<i>Data directed to a specific user. This is data for a specific user such as a statement download, e-mail, or the status of a mortgage application.</i>
Usage Tracking Data	<i>Data about end users. This is data collected about end user behavior (with pre-defined criteria) and sent to the server. This information is available through the Usage Tracking adapter and can be sent to a commercial ad server.</i>

Creating Channel Records

Channel configuration is closely associated with the adapter and datafeed configuration. Adapters are used by channels to fetch data from outside the MoneyScape server; datafeeds manage that data once it has been fetched. This section explains channel records and their association with datafeeds.

Important You must create a channel record for any channel that has a datafeed, that is for any channel that accesses external data. *If a channel doesn't have a datafeed, the channel record is optional.*

A channel can access data through datafeeds to provide Channel Custom Data and User Specific Data to a client (and Usage Tracking Data if desired). Using this metadata, any number of datafeeds can be associated with a channel. They can be grouped together for convenience, and different chunks of data can be sent to different datafeeds in a defined group. Without programming and using a channel's associated datafeed, a client may send the following to adapters:

- Strings
- Strings parsed into tokens based on a delimiter determined by the client
- Strings and tokenized strings to subsets (or groups) of datafeeds (and adapters)
- Objects
- Different objects to different subsets of datafeeds

Additionally, once the data is fetched, it is placed in a file whose name is determined by the metadata. This name can be automatically modified and made unique for multiple, indeterminate numbers of returned files. Any exceptions on the server that can be caught are placed in `UpdateExceptions.txt` and returned for debugging.

As show in the Channel Configurator window, channel records have two parts.

- Channel Data – There is only one set of channel data per channel. This data applies to the channel as a whole.
- Channel Dynamic Data – There are zero to any number of these records per channel, and this data represents a datafeed's (and its adapter's) association with the channel. This is the mechanism for sending Channel Custom Data and User Specific Data to a client (and for receiving Usage Tracking Data).

Channel Data is simply high-level metadata that describes the channel. Channel Dynamic Data is used to associate datafeeds with a channel. *There can be any number of these records per channel and this information is used to associate those datafeeds with a channel.* The datafeed is associated with an adapter and is the mechanism for sending any Custom Channel Data and User Specific Data to a client. Typically, information on the client is added using the `IRequestBuffer` service and is destined for a given datafeed. Once the datafeed gets that information, it can use it to get any Custom Channel Data and User Specific Data, and return it to the client in a file. This information in the configurator manages this association.

Adding, Modifying, and Deleting Records

Use the following procedures to add, modify, or delete records, using standard Windows techniques to navigate in the Configurator windows. Note that there are two ways to add a record: (1) you can add a new record or (2) you can simply rename or copy an existing record.

▼ To add a record

1. Select New from the Configurators menu.
2. Select a hostname.
3. Select a record type.
4. Enter a unique name. (This adds the record to the tree in the left pane.)
5. Configure the record fields as explained below. (If necessary, click Add Record to add additional datafeeds to a channel record.)
6. When done, select Commit from Configurators menu and wait for status message.

▼ To modify a record

1. Right-click on record and choose Rename, Copy, or Delete.
2. Configure the record fields as explained on the following pages. (If necessary, click Add Record to add additional datafeeds to a channel record.)
3. When done, select Commit from Configurators menu and wait for status message.

Channel Name: StartupChannel

Description:
 Startup Channel

General
 Delete Record (CAUTION)
 Data is active
 Require Enrollment
 Debug

Datafeeds for: StartupChannel

Add Record

New	Name to map to	Delete	Active	All Updates	Description	Delimiter	Group Name	Datafeed ID	Disposition
<input type="checkbox"/>	StartupData	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>			DEFAULT	StartupCh...	Always

Channel Data

Channel Dynamic Data

Channel Configurator

Channel Field	Description
Channel Name	Unique name for this channel. <i>This name must be the same as the channel's published name.</i>
Description	Descriptive text for the channel. A string that serves as a description of what the channel is used for.
Delete Record	Permanently deletes the record from the system when checked and committed. <i>Use with caution.</i>
Data is active	Makes the channel's metadata record inactive, and inaccessible, but does not delete the record entirely. This means that only generic channel data is accessible; Custom Channel Data and User Specific data are not.
Require Enrollment	If set, this allows only enrolled users access to the Custom Channel Data and User Specific data this channel can send. An enrollment channel is an example where all users would need access in order to enroll. If only enrolled users had access, no one could enroll.
Debug	Individual channels can be coded to generate debug data. This checkbox determines whether or not that data is written to the <code>java.out</code> log; if not checked, no data is logged. All channels use the same log; right-click on the transmitter name and select View Output to display it.

Channel Field	Description
Name to map to	<ul style="list-style-type: none"> • The Custom Channel Data and User Specific data that a datafeed sends back ultimately becomes a file in the client's channel directory. This variable sets that filename. There are numerous options although the base functionality is usually adequate: • Using a name with no special characters in it (defined below) places the data in a file of the given name. For example, the string: <code>OfxData.txt</code> creates a file with <code>OfxData.txt</code> on the client with the data from the associated datafeed in it. • Having a delimiter character in the name (defined in the delimiter field below) substitutes the chunk of data from the client into the string. For example, if the delimiter character is <code>~</code>, the <code>name to map to</code> is <code>~_News.html</code>, and the chunk of data from the client is <code>meca</code>, then the resulting file is named <code>meca_News.html</code>. In addition, if the chunk of data from the client also contains the delimiter character, it is parsed and inserted into the <code>name to map to</code> string in the corresponding place. This means that the first token replaces the first delimiter, the second replaces the second, etc. For example: Still using the <code>~</code> delimiter example above, if the <code>name to map to</code> is <code>~_News.~</code> and the chunk of data from the client is <code>meca~html</code>, then the file <code>meca_News.html</code> is generated. Note that if there are not enough tokens to be placed in the <code>name to map to</code> string, the last one is repeated. Also note, this is not connected to the token determined on the client using the <code>IRequestBufferService</code>; that token is used to create chunks of data to be sent to an adapter. This delimiter tokenizes the chunk into yet smaller pieces for the purpose of naming the returned file. Finally, this is also different from the delimiter character in the adapters and datafeeds section which is used to create a unique request for an adapter. • A datafeed may return more than one file and this number may vary from call to call. Thus a construct is used to create unique names for each file returned by a single datafeed. If a <code>#</code> character is used in the <code>name to map to</code> string, then the index number of the returned file (starting with zero) is used to replace the <code>#</code> character. So if the <code>name to map to</code> is <code>NewsStory_#.html</code> and there are three files returned, they are named <code>NewsStory_0.html</code>, <code>NewsStory_1.html</code>, and <code>NewsStory_2.html</code>. Note that if a datafeed returns multiple files and this construct is not used, the contents of the last file are returned in the given name. This can also be used for datafeeds returning a single file; the substitution is always be zero. • Leaving the name blank uses the uniquely constructed name sent to the adapter. Note that for some applications, the name an adapter uses may have no meaning in the outside world. • Having a <code>*</code> in the <code>name to map</code> replaces the <code>*</code> with the uniquely constructed name sent to the adapter. Again, for some applications, the name an adapter uses may have no meaning in the outside world.
New	A read-only field that is checked if the record is new.
Delete	Permanently deletes the record from the system when checked and committed. <i>Use with caution.</i>
Active	Makes the association to the datafeed for this channel inactive and inaccessible but does not delete the record entirely. This means that this adapter does not send data to the client. This can be used to stop sending a particular piece of data to a client while not having to stop the entire channel. For example, if a channel sends several different types of financial data to a client and one of the multiple sources for this data goes down, you can set this flag to inactive to stop requests for that specific information while allowing the rest of the channel to update normally. Once the data access is restored, it can be re-activated.

Channel Field	Description
All Updates	Generally, adapters use information from the client to fulfill a request. This is not always the case. There are adapters you may want to call for all channel updates with no arguments, regardless of whether there is any data in the adapter. stream. Setting this flag will make this datafeed called for all updates, with no arguments passed to it. An example of this may be for a datafeed that returns a targeted advertisement, fetched from a commercial ad server. Here, no information is needed from the client to fetch the data. The only needed information is the user's ID which is automatically accessible by the adapter. With that ID, the adapter can make a request from the ad server for the proper data.
Description	Descriptive text for this channel/datafeed relationship.
Delimiter	The delimiter used in name to map to discussed above.
Group Name	<p>The datafeeds available to a channel can be grouped. This means that a given channel can send multiple pieces of data to the server using the <code>IRequestBufferService</code> and target them to different datafeeds. For example, a given channel can supply company news, current stock price, and both yearly and monthly graphs of the stock's price history. If the channel developer wanted to make different requests for each of these services, they could use a group name. For example, the group for the company news datafeed would be <code>NEWS</code>, the quote price datafeed would be <code>QUOTE</code>, and the datafeeds for the two graphs would be <code>GRAPHS</code>. Now, on the client, if the New request was for IBM, this would be added using <code>IRequestBufferService</code> (assume <code>buf</code> is that buffer object):</p> <pre>buf.add("NEWS", "IBM");</pre> <p>Now assume quotes were wanted for IBM, SUNW, and NEC using a space as a chunk delimiter. This would be added as follows.</p> <pre>buf.add("QUOTE", "IBM SUNW NEC", ' ');</pre> <p>Lastly, assume a graph was desired for SUNW:</p> <pre>buf.add("GRAPHS", "SUNW");</pre> <p>This would channel the appropriate chunks of data to the desired datafeeds as determined by the group name. Typically, grouping the datafeeds is unnecessary, so the default group (called <code>DEFAULT</code>) is used and is the target for all requests added without a group. In the above example, assume the same ticker request was always sent to all datafeeds. In this case, if you leave the group name as <code>DEFAULT</code> and use <code>IRequestBufferService</code> without a group name, it sends the chunks of data (delimited by a space) to all datafeeds.</p> <pre>buf.add("IBM SUNW NEC", ' ');</pre> <p>Group names have the following rules:</p> <ul style="list-style-type: none"> • Any datafeed with <code>All Updates</code> checked is always called for during all updates even if there is no data in the request buffer. The group name has no effect here. • By default, the datafeed will have the <code>DEFAULT</code> group name. This means any request from the client that is not explicitly given a group name will go to them. • If a group name is entered for a datafeed overriding <code>DEFAULT</code>, then the datafeed gets only the requests that were explicitly added to the client's request buffer with an exactly matching group name. • If a group name is entered for the datafeed and no exactly matching request is made, the datafeed is not called. • If a client uses a group name for data it added to the request buffer, and there is no corresponding data feed with a matching group name, the request is ignored; it does not go to the default.

Channel Field	Description
Datafeed ID	The name of the datafeed to associate with this channel. This must exactly match the Name field in the adapters and datafeeds section. Once an adapter/datafeed is set up, this is where it gets associated with a channel.
Disposition	<p>The Marimba dispositions for sending the file to the client are explained below.</p> <ul style="list-style-type: none"> If_Needed – Send the byte array to the tuner only if the tuner doesn't already have the data. The data is sent as immediate data which means it will not be cached by a Marimba proxy. This data will be part of the channel's index and will have an effect on optimized update requests. Always – Always send the byte array (file) to the tuner on <i>every</i> request. The data is sent as immediate data which means it will not be cached by a Marimba proxy. This file is not part of the channel's index and will have no effect on optimized updates. Persistent – Treats this byte array as though it were a file. It makes the byte array persistent (for a limited amount of time – at least five minutes) so that the request for this data can be satisfied in a subsequent request by a tuner (or proxy) without plugin intervention. The two dispositions used in all cases are Always and If_Needed. Always sends the file even if an identical file already exists. If_Needed only sends the file if it does not exist (or if it does exist and has changed). Note that if a channel developer deletes a file received from the server, the identical file will be sent during a new update. If If_Needed is used, it will not be re-sent; it will only be re-sent with Always.

Creating Datafeed Records

Datafeed configuration is closely associated with channel configuration. When an adapter is called, it will generally require input (from the client) to uniquely fulfill the request for data. There are three ways you can do this.

- You can send data directly from client to server using the client's `IRequestBufferService`.
- You can insert the information from the client in the form of a string into a pre-defined String on the server to make the request unique.
- You can let the client assemble the data it sends through `IRequestBufferService` into different groups to make channel updates easier.

If you send data directly from the client to the server using the client's `IRequestBufferService`, the data can sent directly to the adapter with no modification. Any kind of data may be sent this way, e.g. a String, byte array, OFX stream, or serialized object.

In other cases, you might want to insert the information from the client in the form of a string into a pre-defined String on the server to make the request unique. This is done by having a **Target Name** (see below) defined for the adapter that contains special characters that are replaced with the information received from the client. The most common example of this is a URL string. For example, if there is an adapter that returns a web home page through HTTP and the special delimiting character is *. The target string might be `http://www.*.com`. Now if the client sent the string `mecasw` via the `IRequestBufferService`, the adapter would receive the string `http://www.mecasw.com`. This facility is particularly useful with the `URLAdapter` or any adapter using URLs to connect to legacy data.

You can also associate multiple adapters with a single channel to let the client assemble the data it sends through `IRequestBufferService` into different groups. On the server, these groups are associated with some subset of the adapters associated with that channel. Thus in a single update for a given channel, the client could send several groups of totally different data to the server, which would then be sent to the correct adapter used to fulfill the request.

By using datafeed configuration, you add an adapter and determine how the data it receives will be managed. Part of the function of channel management is to associate this data (through datafeeds) with a channel update request. This assumes that the adapter is a MoneyScope adapter, or one you developed and put in the `outside_classes` directory.

Important The procedures to add or modify datafeed records are the same as those for channel records. See [Adding, Modifying, and Deleting Records](#) for details.

Datafeed Name: StartupChannel											
Description: Datafeed for Startup Channel											
<input type="checkbox"/> Delete Record (CAUTION) <input checked="" type="checkbox"/> Data is active											
Adapter											
Adapter Class:	com.mecasw.eb.adapters.EnrollmentAdapter										
Target Name:											
Delimiter Character:											
Type of Data	Change Rate										
<input type="checkbox"/> Multiple Chunks	Frequency of change: Always										
	Rate in seconds: 0										
AdapterContext											
<table border="1"> <thead> <tr> <th>Status</th> <th>Consumer Class</th> <th>Type of Data</th> <th>Key</th> <th>Data</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> <td> </td> <td> </td> <td> </td> </tr> </tbody> </table>		Status	Consumer Class	Type of Data	Key	Data					
Status	Consumer Class	Type of Data	Key	Data							

Datafeed Configurator

Datafeed Field	Description
Datafeed Name	A unique name for this datafeed. You can have any number of datafeeds associated with a single adapter. For example, the URLAdapter can have different datafeeds to access different types of data that are available through URLs.
Description	Descriptive text for the datafeed. Typically, a String that serves as a description of what the datafeed is used for.
Delete Record	Permanently deletes the record from the system when checked and committed. Use with caution.
Data is active	Used to make the datafeed inactive and not accessible. Does not delete the record entirely.

Datafeed Field	Description
<p>Adapter Class</p>	<p>The fully qualified classname of the adapter associated with this datafeed. If this datafeed uses adapter called <code>com.mecasw.TestAdapter.class</code>, and the code is placed in the <code>outside_classes</code> directory in <code>outside_classes/com/mecasw/directory</code>, the entry in this field would be <code>com.mecasw.TestAdapter</code>. When the record is saved, this adapter is loaded when this datafeed needs data from outside MoneyScape.</p> <p>Adapters can be handed an <code>AdapterContext</code> object. When an existing record is read, the adapter is queried as to if it wants an <code>AdapterContext</code> handed to it, and if so, what should be in it. If it does want something in <code>AdapterContext</code>, entries show up in the <code>AdapterContext</code> table section of the configurator. The entries can be, in addition to other things, a host adapter or a class. These in turn may also require some <code>AdapterContext</code> entries. Those values also show up.</p>
<p>Target Name</p>	<p>If the adapter receives a String as a target to satisfy this datafeed, this is the String that is either always the same or has some component that is the same. It may contain special delimiter characters that are replaced by chunks of data that the client sends.</p>
<p>Delimiter Character</p>	<p>If the <code>TargetName</code> string is set, and you wish to insert some string(s) from the client into it via the <code>IRequestBufferService</code> to make the request unique, they will be replaced with the chunk of data from the client if any of the characters chosen are in the <code>TargetName</code>. For example, assume there is an adapter that returns a web home page through HTTP and the special delimiter character is <code>*</code>. The <code>target string</code> might be <code>http://www.*.com</code>. Now if the client sent the string <code>meca</code> via the <code>IRequestBufferService</code>, the adapter would receive the string <code>http://www.meca.com</code>.</p> <p>You can have multiple substitutions in a given string with multiple values. The chunk of data from the client is broken up by the same delimiter character. Using the same web home page example, the <code>TargetString</code> could be: <code>http://www.*.*</code>, and if the chunk of data from the client was <code>meca*com</code>, the string would expand to <code>http://www.meca.com</code>. If the string from the client was <code>pbs*org</code>, the <code>TargetString</code> would expand to <code>http://www.pbs.org</code>.</p>
<p>Multiple Chunks</p>	<p>Generally, data is returned as a single thing, e.g. a web page, a serialized object, or an OFX file. However, you can have an adapter return a bundle of things, e.g. news stories for a particular company being sent as web pages. By checking this option, multiple web pages are returned as separate files with one call to the adapter (see the javadocs for <code>com.mecasw.eb.datafeed.DataBundle</code> for details on implementing this programmatically.)</p>

Datafeed Field	Description
Frequency of change	<p>The MoneyScope server can cache data locally that was previously fetched from adapters. This can significantly reduce the time and processing needed to send data to end users. For example, you may want to cache the current mortgage interest rate that changes only once every 24 hours. If there were 10,000 hits a day for that information without caching, each hit would require a request through an adapter to the legacy system. With caching, there is only one request to the legacy system per day.</p> <p>Caching is especially useful for (1) reducing manipulation of large pieces of data; (2) data requiring a process-intensive operation to acquire (it reduces the response time to client and increases efficiency of the server and legacy systems); (3) highly requested data (reduces the sheer number of hits to the legacy system); (4) infrequently changing data. This field controls if and how the data is cached. There are five caching options:</p> <ul style="list-style-type: none"> • Always – Always get it from the adapter; this data is not cached. • Never Expire – Always get it from cache. If there no data in the cache, get it from the adapter. The data never expires. If not found either in cache or from the adapter, null is returned. • Scheduled – Never get it from the adapter. Similar to Never Expire except the data in cache does change and is placed there by some external process. • Sometimes – Always check cache. If found, see if the data is stale by checking Rate in seconds setting. If stale, get from adapter. An exception is thrown if the data is stale and new data can't be retrieved from the adapter. • If Available – Always check cache. Similar to Sometimes, except if stale, use cache anyway. An exception is thrown only if no data is found.
Rate in seconds	<p>If you cache the data with Sometimes or If available above, this field determines how long the data is considered fresh. So, if data is fetched from an adapter and is set for caching, and the change rate is one hour, any subsequent request from a client for that data will be satisfied from cache for the next hour. The first request for the data after it becomes stale will prompt the server to get fresh data from the adapter and cache it. Again, for the next hour, the data will come from cache. If desired, the starting or creation time of the data can be set by the adapter developer. If not, by default, the creation time is when the data is retrieved from the adapter.</p>
Adapter Context	<p>When an existing record is read, the adapter class is queried to find out if it requires any data be inserted in its AdapterContext. For each piece of data required, there will be one line in the table. The information here is saved only if the data field has a value. The data field may contain other classes used by the adapter. These may also require adapterContext entries. They are also queried to see if they require any entries, and if so, they are listed here too. All fields are read only except for the data field which may be read/write. See the Developing Adapters section of the <i>Channel Developer Guide</i> for more information.</p>
Status	<p>For any existing records (those where the data field has a value), the adapter is queried to check the status of this entry. There are three possible values:</p> <ul style="list-style-type: none"> • CONFIRMED – The query to the adapter or class confirmed that this entry is in fact used. • UNCONFIRMED – The adapter or class could not be accessed by the server so this entry could not be confirmed as being used. • UNREACHABLE – The adapter or class does not use this entry. This means that if an adapter had used an AdapterContext entry called URL, and there was a record entry here for it, and now the adapter was changed and no longer uses URL, this would cause the flag UNREACHABLE to be displayed. If it is true that this will no longer be needed, delete the data value and the record will be deleted.

Datafeed Field	Description
Consumer Class	As discussed, the adapter may be handed other classes. These other classes may also require adapter context records which will be displayed here. This field shows the name of the class that actually uses this entry: the consumer of this record.
Type of Data	<ul style="list-style-type: none"> <code>hostadapter</code> – Often there are two parts to an adapter, a generic part which provides some basic functionality or services, and a host-specific part for connecting to and communicating with something outside the MoneyScape server. Through this metadata, a single generic part can be handed different host-specific parts in its <code>AdapterContext</code>. This type is reserved for classes used by adapters for host-specific activities. An example is the <code>FinancialAdapter</code>. Here the actual adapter provides basic generic financial information. Once processed, it is handed to the <code>hostadapter</code> it received in its <code>AdapterContext</code>. This can be a URL connector, a socket connector, or some custom host communication piece. An instantiation of this is handed to the consumer class. <code>object</code> – Any other object needed by the adapter. An instantiation of this is handed to the consumer class. <code>userid</code> – This is the userid of the client on whose behalf this adapter instance is executing. This value is set by the server before calling the adapter, and therefore this will render the data field non-editable. <code>PropertyFile</code> – A <code>java.util.Property</code> class is instantiated and placed in the <code>AdapterContext</code>. The name of the property file is specified in the <code>Data</code> field. The file must reside in <code>\$MECA_HOME/etc</code> <code>String</code> – A <code>java.lang.String</code> whose value is the string entered in the <code>Data</code> field. <code>timestamp</code> – This is set by the adapter and read by the server. It represents the time that the data returned by the adapter was created. This will render the data field non-editable.
Key	The adapter or class must specify a key name for each piece of data. This is that key name.
Data	This is where the administrator enters the information for this entry. For example, if this is a <code>hostadapter</code> , this field may be <code>com.mecasw.util.HTTPConnector</code> . If this were a <code>String</code> and key was a URL, this may be <code>http://www.mecasw.com</code> .

Configuration Examples

The following examples show the process needed to develop some typical adapters. The following high level steps are required to accomplish this, although many of these steps can be done concurrently. See the *Channel Developer Guide* for a wider discussion of adapter development.

1. Define the desired client functionality. (This is what the end user sees when they subscribe to a MoneyScape channel.)
2. Define the client and adapter functionality, and what the client and adapter send to each other (strings, serialized objects, arrays of objects, etc.).
3. Write the adapter code (if a pre-built adapter is not adequate).
4. Using the Server Administrator, add the adapter to the server using the datafeed configurator.
5. Using the Server Administrator, add a channel record for the channel with the channel configurator. This associates the datafeed from above with the channel.

6. Write the actual Java code for the channel.
7. Publish and test the channel.

AnswerMyQuestion Channel and Adapter – Simple Example

In this example, the client channel submits a question to the MoneyScape server, and that question is sent to an adapter for an answer. The response is returned in the file `Answer.txt`. There are many ways to do this, but for this example, the client will add a String to the `IRequestBufferService`, and the adapter will be handed this String which it will answer by sending a String back. For simplicity, only two possible questions are asked, “What is the Date and Time?” or “What is your Name?”. This completes the first two steps defined above.

Next write the adapter. Normally, an adapter will use the information it received to get information from outside the MoneyScape server to fulfill the request. In this case, the adapter (`com.test.adapters.AnswerQuestionAdapter.java`) will only do the following.

```
public Object process(Object obj) throws AdapterException
{
    String question = (String)obj;

    if(question.equals("What is the Date and Time?"))
        return(new String("The Date and Time is " + new Date()));
    else // What is your name?
        return("My name is QuestionAnswerAdapter");
}
```

When done, put the adapter in the `outside_classes` directory on the server and use the Server Administrator to add it. First we configure the datafeed and adapter. We create a new datafeed record, overriding the defaults, as follows.

Datafeed Record	
Datafeed Name	QuestionAndAnswer
Description	A demonstration adapter to show request and response functionality.
Data is active	Check this to make the adapter available.
Adapter Class	<code>com.test.adapters.AnswerQuestionAdapter</code>
Multiple chunks	No
Frequency of Change	Always. For this example, we want the data to always come from the adapter.

Next, create a new channel record, overriding the defaults as follows.

Channel Record	
Datafeed Name	AnswerMyQuestion
Description	A demonstration channel used to show how information flows from client to server and back.
Data is active	Yes

Channel Record	
Require Enrollment	No. For this example, don't care.

Next, for the channel, we create one new Channel Dynamic data record, associating the adapter/datafeed with this channel.

Channel Dynamic Data	
Name to Map to	Answer.txt
Active	Yes
All updates	No
Description	Datafeed used to answer the question coming from the client.
Delimiter	Leave blank.
Group Name	Leave as DEFAULT
DataFeed Id	QuestionAndAnswer. The Name used when creating the datafeed/adaptor information.
Disposition	ALWAYS. We will always send the file.

This is all you need. The server is ready. If the client, using the `IRequestBufferService`, does this:

```
buf.add("What is the Date and Time?");
```

A file called `Answer.txt` is returned with the string:

```
"The Date and Time is Tuesday December 7, 1999 07:21:02".
```

If the client sends this:

```
buf.add("What is your Name?");
```

A file called `Answer.txt` is returned with the string:

```
"My name is QuestionAnswerAdapter".
```

AnswerMyQuestion Channel and Adapter – Complex Example

The first example asked a single question. If the client adds more than one question in a single update, only the answer to the last is returned to the client. This is because the metadata is set up so that the response to the question goes into a single file called `Answer.txt`. Each time a datafeed returns a response to the plugin, it overwrites the previous file with the contents of the new file. In the next example, we will allow multiple questions. There are several potential ways to do this and the following two examples are discussed:

- One file returned for each answer
 - Set up the metadata so each answer is returned in a separate file. (Several variations are available.)
 - Send multiple questions as a single chunk and have the adapter parse them and send the response in a `DataBundle` class.
- One file returned with all answers

- Set up the delimiters so that two questions are added as a single chunk and the adapter parses them and puts all responses in a single file.
- Instead of adding a single String on the client, add the questions to another object, e.g. a Vector, and send that. The adapter extracts each question from the Object and adds each response to its return String, sending that String when done.
- Either of the above, but instead of sending the response as a String, send the response as a serialized Object that would be understood by the client and that would house the responses.

One File Returned for Each Answer

The first method only requires a metadata change and one of several naming options that are available. This only shows those fields different than the first example. First, we set up the metadata so that the returned files will be the same as the question asked.

Datafeed Record	No modifications necessary
Channel Record	No modifications necessary
Channel Dynamic Data	Name to map to: Leave blank

This tells the plugin to use the unique identifier that was sent to the adapter to fulfill the request. The unique identifier sent to the adapter is the question (in the form of a string).

If the client, using the `IRequestBufferService`, does this:

```
buf.add("What is the Date and Time?");
```

A file called `What is the Date and Time?` is returned with the string:

```
"The Date and Time is Tuesday December 7, 1999 07:21:02".
```

If the client sends this:

```
buf.add("What is your Name?");
```

A file called `What is your Name?` is returned with the string:

```
"My name is QuestionAnswerAdapter".
```

Here is a variation.

Datafeed Record	No modifications necessary
Channel Record	No modifications necessary
Channel Dynamic Data	Name to map to: *.txt

This tells the plugin to use the unique identifier that was sent to the adapter in place of the `*` character in the `Name to map to`. The unique identifier is sent to the adapter if the question is in the form of a string.

If the client, using the `IRequestBufferService`, and does this:

```
buf.add("What is the Date and Time?");
```

A file called `What is the Date and Time?.txt` is returned with the string:

"The Date and Time is Tuesday December 7, 1999 07:21:02".

If the client sends this:

```
buf.add("What is your Name?");
```

A file called `What is your Name?.txt` is returned with the string:

```
"My name is QuestionAnswerAdapter".
```

The next option for a multiple file return is to have the adapter return a `DataBundle`. The `DataBundle` lets an adapter return any number of distinctly different pieces with one call and return. The client code concatenates each question into one request, and those concatenated questions represent one chunk of data. The questions are parsed by the adapter; each question is answered and that answer is added to the `DataBundle`, and the `DataBundle` is returned. The answers are placed in files called `Answer0.txt`, `Answer1.txt`, `Answer2.txt`, etc. Again, only the changes from the initial simple example are shown:

Datafeed Record	Multiple chunks: Yes
Channel Record	No changes.
Channel Dynamic Data	Name to Map to: Answer#.txt

This tells the plugin to replace the # with the index number for that particular piece of data from the adapter. Assume the adapter parses the questions by the ? character. If the client, using the `IRequestBufferService`, does this:

```
buf.add("What is the Date and Time? What is your Name?");
```

A file called `Answer0.txt` is returned with the string:

```
"The Date and Time is Tuesday December 7, 1999 07:21:02".
```

And a file called `Answer1.txt` is returned with the string:

```
"My name is QuestionAnswerAdapter".
```

One File Returned for All Answers

None of these variations requires metadata changes from the initial simple example. All changes are code changes depending on the desired functionality. The coding details are not shown but it demonstrates the versatility of what can be sent through the MoneyScape server.

First, as in the last example above, the client concatenates multiple questions into one request, and the adapter parses them into separate questions. In this case it answers each question, concatenates the answers, and sends back one response. That response becomes `Answer.txt`.

In the next example, a serialized object is added on the client, and the adapter de-serializes it, extracts the information, and responds by concatenating the strings together like above. The client code to add the object might look like this:

```
Vector v = new Vector()
v.addElement("What is the Date and Time?");
v.addElement("What is your Name?");
buf.add(MECAUtil.objectToByteArray(v));
```

Lastly, the adapter could have inserted the answers in an Object like a Vector, and serialized and returned it, instead of sending concatenated strings. The server would place the serialized Object in the file, and the client would de-serialized it and extract the information accordingly.

AnswerMyQuestion – Using Cache

In previous examples, all requests went directly to the adapter without using cache. The next two examples demonstrate how to use cache and also how dissimilar pieces of data can be grouped together. Looking at the answers to the possible questions, the answer to “What is the Date and Time?” will always change and can’t be cached. The answer to “What is your Name?” never changes and can be cached.

To do this, we set up two different datafeeds, one for the Date and Time, and one for Name. They will both use the same adapter but will manage the data that is returned differently. In the channel record, we will use grouping so the right question goes to the right adapter. The datafeed/adapter records are essentially the same as in the original simple example with some minor changes. Listed below are all modified fields that differ from the default.

This is the adapter/datafeed entry for the `DateTime` datafeed:

Datafeed Record	
Name	<code>DateTimeQuestionAdapter</code>
Description	This will return the <code>DateTime</code> when asked.
Data is active	Check this to make the adapter available.
Adapter Class	<code>com.test.adapters.AnswerQuestionAdapter</code>
Multiple chunks	No
Frequency of Change	Always. We want the question to always go to the adapter.

This is the adapter/datafeed entry for the `Name` datafeed:

Datafeed Record	
Name	<code>NameQuestionAdapter</code>
Description	This will return the name of the adapter when asked.
Data is active	Check this to make the adapter available.
Adapter Class	<code>com.test.adapters.AnswerQuestionAdapter</code>
Multiple chunks	No
Frequency of Change	Never Expire. Always use cache. We don’t need this information from the adapter unless not in cache. This means the first time asked, it will not be in cache so the adapter will be asked the question. The response will be put in cache for answering subsequent calls. All subsequent requests will not require a call to the adapter.

Next, create a new channel record. This is exactly the same as the simple example shown earlier. Listed below are the settings to override the defaults:

Channel Record	
Name	AnswerMyQuestion
Description	A demonstration channel used to show how information flows from client to server and back.
Data is active	Yes
Require Enrollment	No. For this example, don't care.

Next, for the channel, we create one new channel dynamic data record for each of the two datafeeds we want to access. They are similar to the simple example, except that we now use group names and associate them with the appropriate datafeed. First the `DateTime` record:

Channel Dynamic Data	
Name to Map to	DateTimeAnswer.txt
Active	Yes
All updates	No
Description	Datafeed used to answer the question coming from the client.
Delimiter	Leave blank.
Group Name	<code>DATETIME</code> We want all data coming from the client associated with the <code>DATETIME</code> group to come to this adapter.
DataFeed Id	<code>DateTimeQuestionAdapter</code> The Name of the adapter/datafeed to go to.
Disposition	Always. Always send the file.

Next the `Name` record:

Channel Dynamic Data	
Name to Map to	NameAnswer.txt
Active	Yes
All updates	No
Description	Datafeed used to answer the question coming from the client.
Delimiter	Leave blank
Group Name	<code>NAME</code> . We want all data coming from the client associated with the <code>NAME</code> group to come to this adapter.
DataFeed Id	<code>NameQuestionAdapter</code> The Name of the adapter/datafeed to go to.
Disposition	<code>ALWAYS</code> Always send the file.

The questions are now grouped on the client side when making requests. Note that this is a simple example. In a more complex example, any number of chunks of data (e.g. questions) could be sent to each of the groups. Note that since there is no adapter set up as the `DEFAULT`,

if no group name is specified, the request would be discarded once it arrived at the server. Here is the client piece:

If the client, using the `IRequestBufferService`, does this:

```
buf.add("What is the Date and Time?", "DATE");
```

A file called `DateTimeAnswer.txt` is returned with the string:

```
"The DateTime is Tuesday December 7, 1999 07:21:00".
```

The request would have gone directly to the adapter without using cache.

If the client sends this:

```
buf.add("What is your Name?", "NAME");
```

A file called `NameAnswer.txt` is returned with the string:

```
"My name is QuestionAnswerAdapter".
```

If this was the first request to the server for this information, the server would have checked cache, and found that there was no data; then it would have sent the question to the adapter which would have returned the answer. The answer would have been placed in cache so that subsequent requests would have found the result without contacting the adapter.

AnswerMyQuestion – Using Cache with Expiring Data

This example is similar to the cache example above but shows how to set up data that is cached and then expires. To do this, assume that the question is changed to “What is the Date?” (not the date and time). The date is the same for 24 hours, so we set up the datafeed to keep and use the cached data for 24 hours after which it expires and must be re-fetched from the adapter. This means that the adapter would only be hit once per day, for any number of requests from that datafeed for that information.

When data is returned from an adapter, it is associated with a creation time. If the data is set to expire, this time is used to set a start point for determining when the data is stale. *By default, if the adapter does not explicitly set a creation time, the time the record was fetched from the adapter is used.* In this case, the adapter needs to explicitly set that time.

If the first request for the date arrives at 1:00 AM, the default creation time would not be correct because it would be one hour after the date change; the data would be stale in 23 hours rather than 24. So the adapter would set the creation time to the current time minus the number of total seconds past midnight. This would be done by using the `AdapterContext` that is available to all adapters and is used to get and set required information. (See the *Channel Developer Guide* for more details.) The changes would consist of writing the adapter, and minor differences in the adapter/datafeed entry from the above example.

This is the adapter/datafeed entry for the `Date` datafeed:

Datafeed Record	
Name	DateQuestionAdapter
Description	This returns today’s date when asked. The date is good for 24 hours and the cache is set for that.
Data is active	Check this to make the adapter available.

Datafeed Record	
Adapter Class	<code>com.test.adapters.AnswerQuestionAdapter</code>
Multiple chunks	No
Frequency of Change	Sometimes. This means to check cache. If there is data in cache, determine if it is fresh enough to use. This is determined by the Rate in Seconds.
Rate in Seconds	86400 This is the number of seconds in 24 hours. Use if required by the Frequency of Change. This means the data changes every 86400 seconds, so that 86400 seconds after the creation time of the data in cache, the adapter would need to be accessed again.

All other factors are essentially the same. The same adapter can be used for any of these types of caching, with no prior knowledge of whether or not the data is cached.