Tamino

Tamino Manager User Documentation

Version 10.7

May 2021

WEBMETHODS

This document applies to Tamino Version 10.7 and all subsequent releases.

Document ID: INS-MANAGER-107-20210510

Table of Contents

Tamino Manager

Welcome to the documentation for the Tamino Manager, the administration component for the Tamino XML Server. This document is intended for database administrators and for application developers.

Different from previous versions Tamino, starting with version 10.1, does not anymore include a graphical user interface for administration purposes. Instead the **inoadmin** tool is extended to provide full administration powers.

This documentation contains the following sections:

Introduction to inoadmin	This is a general overview of the command line utility **inoadmin**.
inoadmin Features	This section provides an introduction to the main features of the the command line utility **inoadmin**.
Tamino Security	This document contains information about Tamino authentication and authorization.
Administration Considerations	This section provides general information about Tamino space management and problem reporting.

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at **https://documentation.softwareag.com**.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at **https://empower.softwareag.com/**.

You can find product information on the Software AG Empower Product Support website at **https://empower.softwareag.com**.

To submit feature/enhancement requests, get information about product availability, and download products, go to **Products**.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the **Knowledge Center**.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at **https://empower.softwareag.com/public_directory.aspx** and give us a call.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at **http://techcommunity.softwareag.com**. You can:

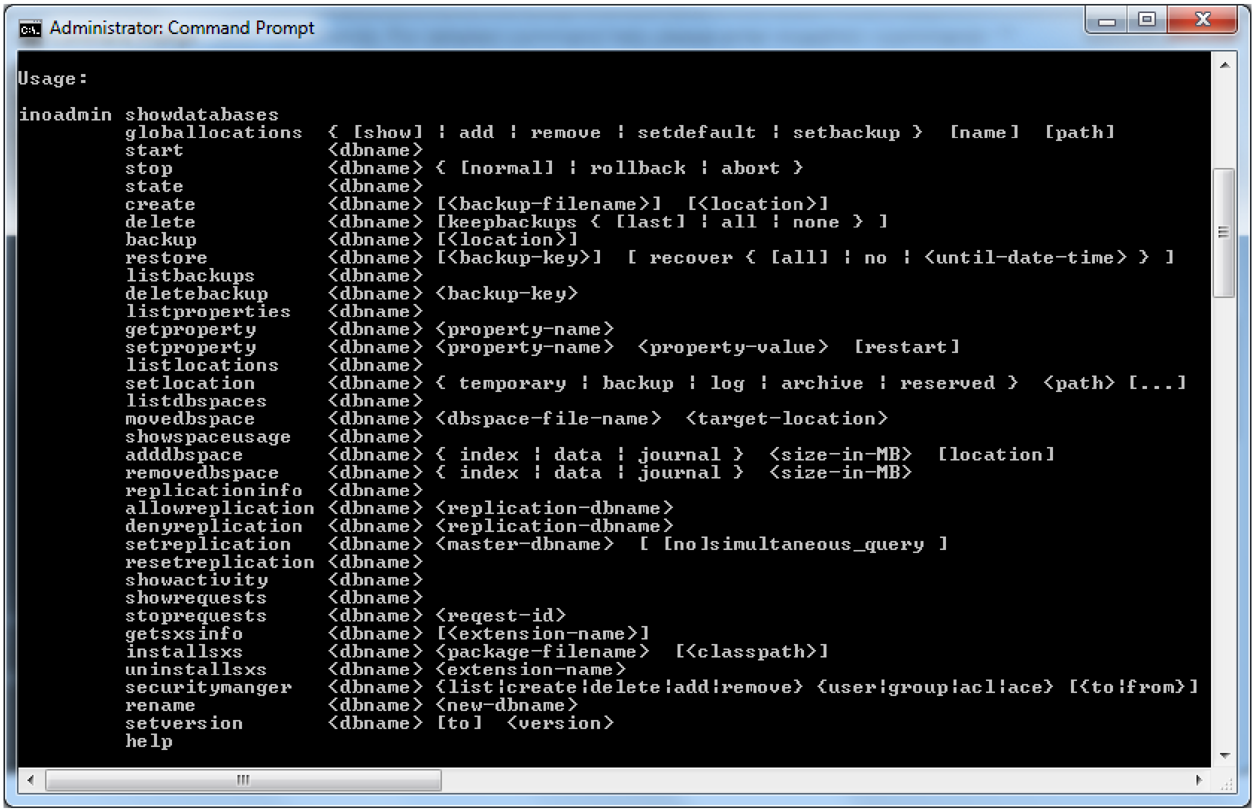
- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Introduction to inoadmin

Tamino provides an administration tool that is implemented as a command line utility. The tool runs in the folder "<INSTALL_ROOT>/Tamino/v107/bin". To set up the administration environment first run the command "ino_setenv". Then, just calling "inoadmin" produces a list of available administration commands. For detailed command help please enter inoadmin <command> "?".



```
Usage:

inoadmin showdatabases
         globallocations  { [show] | add | remove | setdefault | setbackup }  [name]  [path]
         start            <dbname>
         stop             <dbname> { [normal] | rollback | abort }
         state            <dbname>
         create           <dbname> [<backup-filename>]  [<location>]
         delete           <dbname> [keepbackups { [last] | all | none } ]
         backup           <dbname> [<location>]
         restore          <dbname> [<backup-key>]  [ recover { [all] | no | <until-date-time> } ]
         listbackups      <dbname>
         deletebackup     <dbname> <backup-key>
         listproperties   <dbname>
         getproperty      <dbname> <property-name>
         setproperty      <dbname> <property-name>  <property-value>  [restart]
         listlocations    <dbname>
         setlocation      <dbname> { temporary | backup | log | archive | reserved }  <path> [...]
         listdbspaces     <dbname>
         movedbspace      <dbname> <dbspace-file-name>  <target-location>
         showspaceusage   <dbname>
         adddbspace       <dbname> { index | data | journal }  <size-in-MB>  [location]
         removedbspace    <dbname> { index | data | journal }  <size-in-MB>
         replicationinfo  <dbname>
         allowreplication <dbname> <replication-dbname>
         denyreplication  <dbname> <replication-dbname>
         setreplication   <dbname> <master-dbname>  [ [no]simultaneous_query ]
         resetreplication <dbname>
         showactivity     <dbname>
         showrequests     <dbname>
         stoprequests     <dbname> <reqest-id>
         getsxsinfo       <dbname> [<extension-name>]
         installsxs       <dbname> <package-filename>  [<classpath>]
         uninstallsxs     <dbname> <extension-name>
         securitymanger   <dbname> <list|create|delete|add|remove> <user|group|acl|ace> [<to|from>]
         rename           <dbname> <new-dbname>
         setversion       <dbname> [to]  <version>
         help
```

3 inoadmin Features

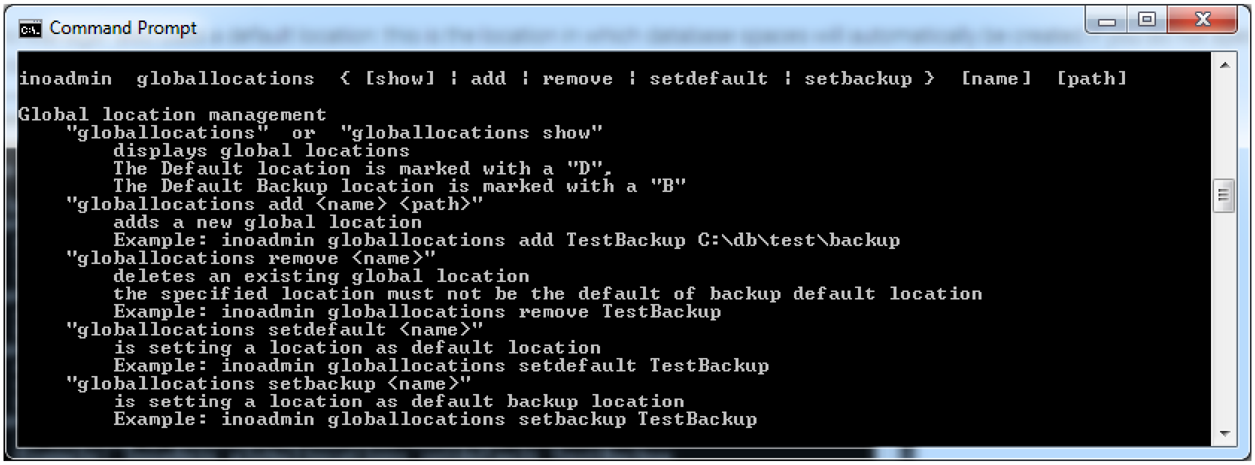
This documentation covers the following topics:

Administration of Database Locations

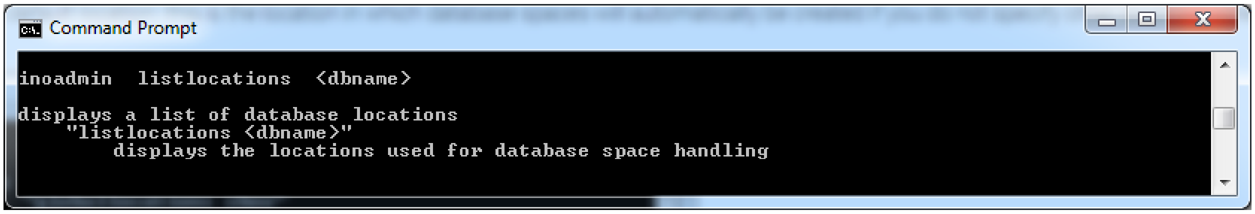
In order to specify physical storage on a node, the Tamino Manager uses the concept of database locations. Database locations define where the Tamino database spaces are stored. The Tamino Manager maps these database locations (which can also be thought of as logical names) to physical location paths.

Let us suppose that a database administrator has a directory called *C:\Tamino Database Locations\Backups\mydbbackup*, in which he plans to store backup copies of his database `mydb`. Using the **Locations** object, he can assign the database location name of `mydbBackup` to this directory. Once this has been done, he can administrate the files via the location name without having to remember where they are physically located.

The Tamino Manager also uses a default location: this is the location in which database spaces will automatically be created if you do not specify other locations when creating a new database. The initial setting uses a location as default location which is created during installation.

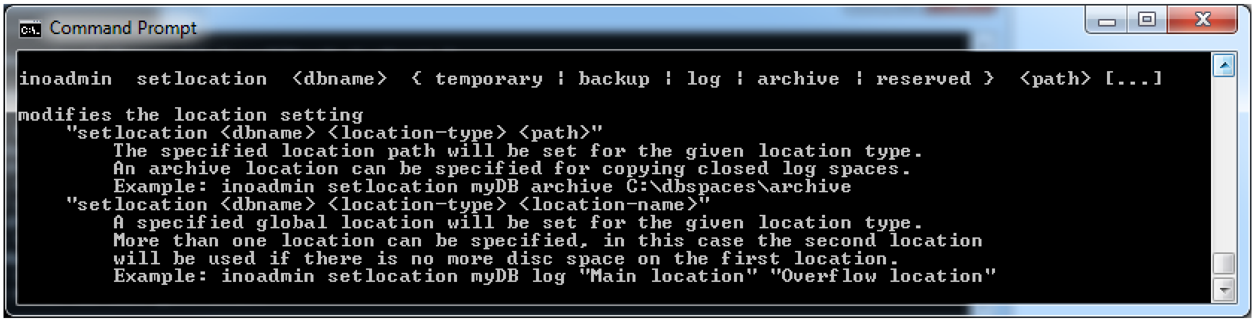


```
inoadmin  globallocations  { [show] | add | remove | setdefault | setbackup }  [name]  [path]

Global location management
    "globallocations"  or  "globallocations show"
        displays global locations
        The Default location is marked with a "D",
        The Default Backup location is marked with a "B"
    "globallocations add <name> <path>"
        adds a new global location
        Example: inoadmin globallocations add TestBackup C:\db\test\backup
    "globallocations remove <name>"
        deletes an existing global location
        the specified location must not be the default of backup default location
        Example: inoadmin globallocations remove TestBackup
    "globallocations setdefault <name>"
        is setting a location as default location
        Example: inoadmin globallocations setdefault TestBackup
    "globallocations setbackup <name>"
        is setting a location as default backup location
        Example: inoadmin globallocations setbackup TestBackup
```



```
inoadmin  listlocations  <dbname>

displays a list of database locations
    "listlocations <dbname>"
        displays the locations used for database space handling
```

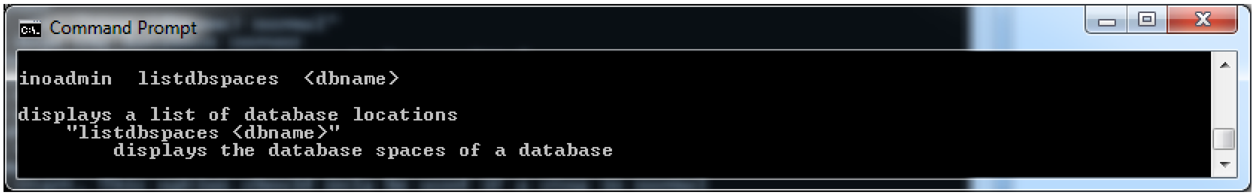
The **setlocation** command defines locations specified by their path as locations for specific purposes to be used by the specified database. Possible location usages are temporary, backup, log, archive, and reserved.



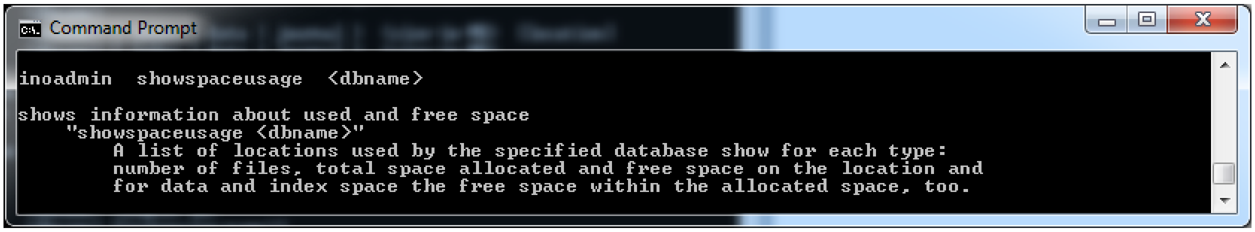
```
inoadmin  setlocation  <dbname>  { temporary | backup | log | archive | reserved }  <path> [...]

modifies the location setting
    "setlocation <dbname> <location-type> <path>"
        The specified location path will be set for the given location type.
        An archive location can be specified for copying closed log spaces.
        Example: inoadmin setlocation myDB archive C:\dbspaces\archive
    "setlocation <dbname> <location-type> <location-name>"
        A specified global location will be set for the given location type.
        More than one location can be specified, in this case the second location
        will be used if there is no more disc space on the first location.
        Example: inoadmin setlocation myDB log "Main location" "Overflow location"
```

When specifying log locations, multiple selections are interpreted as follows: The locations will be used in the same order as in the list. Tamino will use the log locations alternately within a session when a log space switch occurs. When a new session is started, the first available location is used. In contrast, for other location types the second location will only be used when the first location cannot be written to,

Further commands are available to handle database space issues.



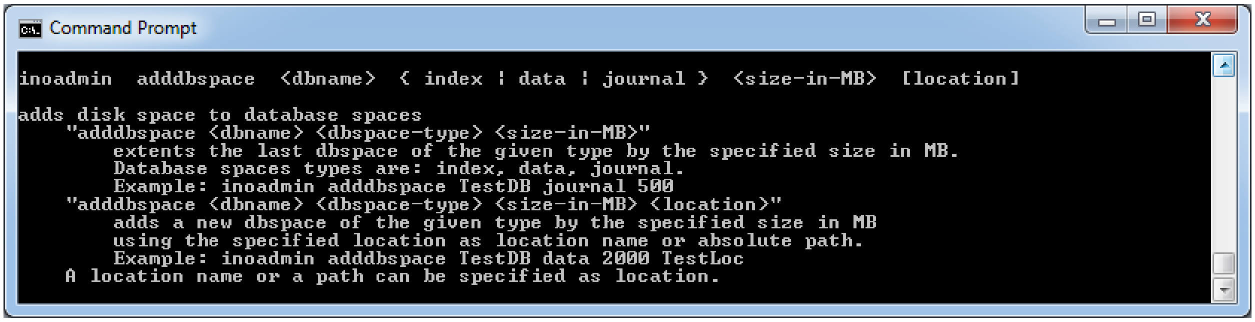
```
inoadmin  listdbspaces  <dbname>

displays a list of database locations
    "listdbspaces <dbname>"
        displays the database spaces of a database
```



```
inoadmin  showspaceusage  <dbname>

shows information about used and free space
    "showspaceusage <dbname>"
        A list of locations used by the specified database show for each type:
        number of files, total space allocated and free space on the location and
        for data and index space the free space within the allocated space, too.
```

The **adddbspace** command adds new space of the types index, data and journal to an existing database in order to allow for database growth. The database to which the new database space is to be added can be either active (running) or inactive (stopped). The command can be used to create a new, separate, space of a specific kind to a database, as well as to expand an already existing space.

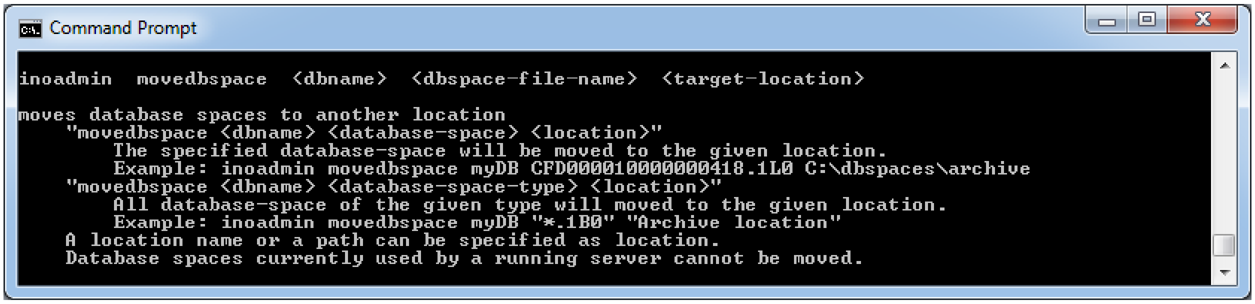
Note: If the property `autoexpand` is enabled, Tamino allocates the needed database spaces by itself.



```
inoadmin  adddbspace  <dbname>  { index | data | journal }  <size-in-MB>  [location]

adds disk space to database spaces
    "adddbspace <dbname> <dbspace-type> <size-in-MB>"
        extents the last dbspace of the given type by the specified size in MB.
        Database spaces types are: index, data, journal.
        Example: inoadmin adddbspace TestDB journal 500
    "adddbspace <dbname> <dbspace-type> <size-in-MB> <location>"
        adds a new dbspace of the given type by the specified size in MB
        using the specified location as location name or absolute path.
        Example: inoadmin adddbspace TestDB data 2000 TestLoc
    A location name or a path can be specified as location.
```

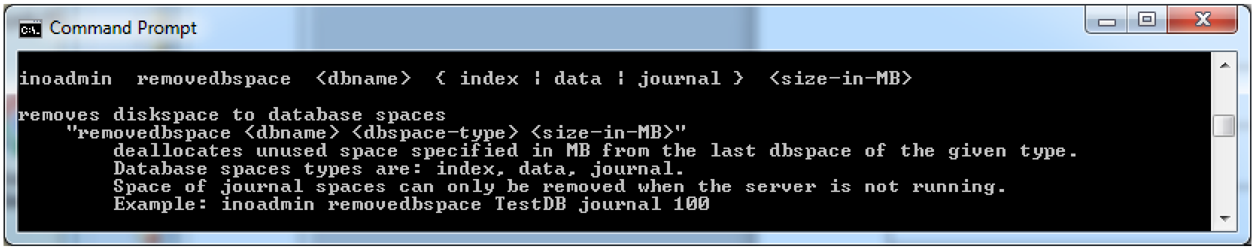
The **movedbspace** command is used to move a database space (data, index, journal or journal overflow) to a new physical location. The database that contains the space to be moved must be inactive (stopped).

For Journal Overflow Space, this is the only command that is available. Journal overflow is automatically created or extended if it is necessary, and it will also be automatically removed or decreased during server shutdown.



```
inoadmin  movedbspace  <dbname>  <dbspace-file-name>  <target-location>

moves database spaces to another location
    "movedbspace <dbname> <database-space> <location>"
        The specified database-space will be moved to the given location.
        Example: inoadmin movedbspace myDB CFD0000100000000418.1L0 C:\dbspaces\archive
    "movedbspace <dbname> <database-space-type> <location>"
        All database-space of the given type will moved to the given location.
        Example: inoadmin movedbspace myDB "*.1B0" "Archive location"
    A location name or a path can be specified as location.
    Database spaces currently used by a running server cannot be moved.
```

The **removedbspace** command is used to delete the most recently-added index, data or journal space from a database. The database that contains the space to be deleted must be inactive (stopped).



```
inoadmin  removedbspace  <dbname>  { index | data | journal }  <size-in-MB>

removes diskspace to database spaces
    "removedbspace <dbname> <dbspace-type> <size-in-MB>"
        deallocates unused space specified in MB from the last dbspace of the given type.
        Database spaces types are: index, data, journal.
        Space of journal spaces can only be removed when the server is not running.
        Example: inoadmin removedbspace TestDB journal 100
```

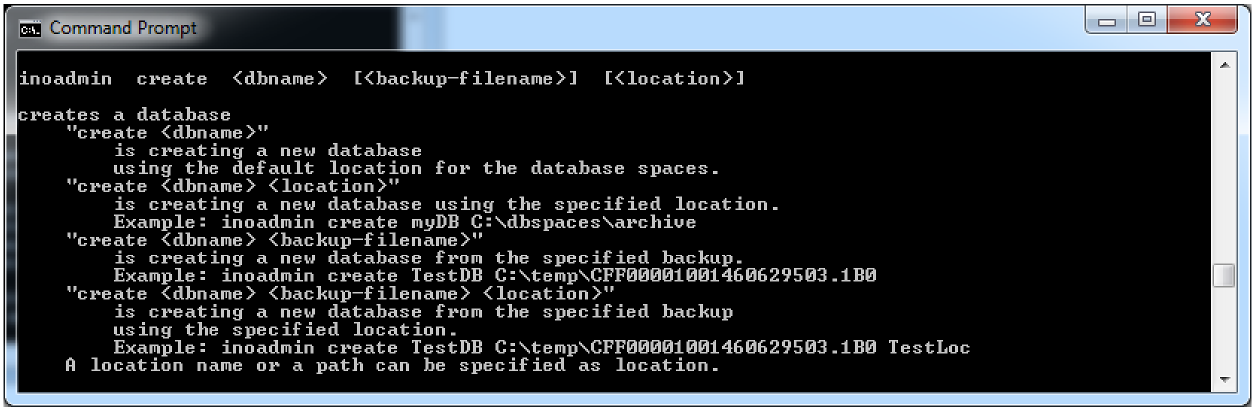
For further information about database spaces, please refer to the page about **Administration Considerations**.

Database Handling

inoadmin provides commands for starting and stopping a database identified by name, as well as a command to ask a database for its state. Furthermore, databases can be created, deleted, and renamed.

Databases are created using **create**. **create** can be used to create a new, empty, database as well as to provide an exact copy (at a given status) of an existing database, for example on another machine or when performing a database restore operation.

Note: A database can only be created from a backup on the hardware architecture/operating system on which the backup was made.

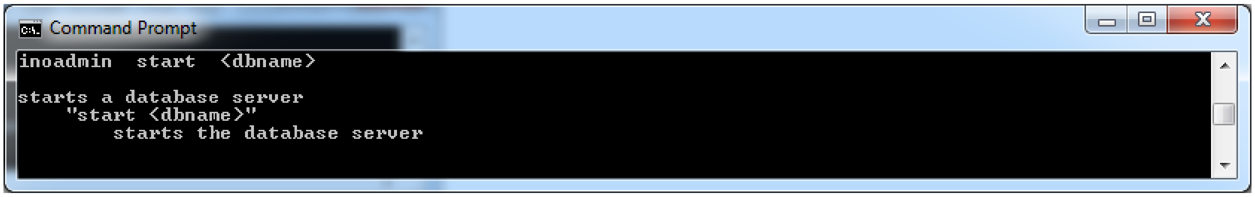


```
inoadmin  create  <dbname>  [<backup-filename>]  [<location>]

creates a database
    "create <dbname>"
        is creating a new database
        using the default location for the database spaces.
    "create <dbname> <location>"
        is creating a new database using the specified location.
        Example: inoadmin create myDB C:\dbspaces\archive
    "create <dbname> <backup-filename>"
        is creating a new database from the specified backup.
        Example: inoadmin create TestDB C:\temp\CFF00001001460629503.1B0
    "create <dbname> <backup-filename> <location>"
        is creating a new database from the specified backup
        using the specified location.
        Example: inoadmin create TestDB C:\temp\CFF00001001460629503.1B0 TestLoc
    A location name or a path can be specified as location.
```

The database name can be up to a maximum of 32 characters in length. The following characters may not be used: SPACE , ; ! \ = # . [ ] " % ' / ^ ` ? *

When you create a database using the Create Database From Backup command, Tamino creates the database in such a way that any fragmentation that might have been present in the original database is removed. The defragmentation can improve the efficiency of the database's internal indexing mechanism and also reduce the overall physical disk requirements of the database spaces.

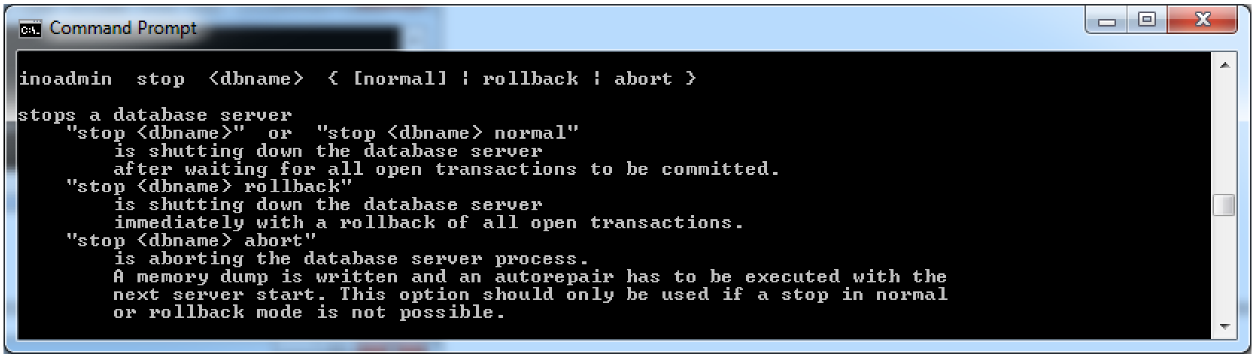
The **start** command starts a dabase server identified by its name.



```
inoadmin  start  <dbname>

starts a database server
    "start <dbname>"
        starts the database server
```

In general, you can only start a Tamino database on the platform on which it was created; for example, you cannot start a Tamino database on Solaris if the database was created with Tamino on Windows. This is because the format of data generally differs from one operating system to another.

The **stop** command stops a dabase server identified by its name.



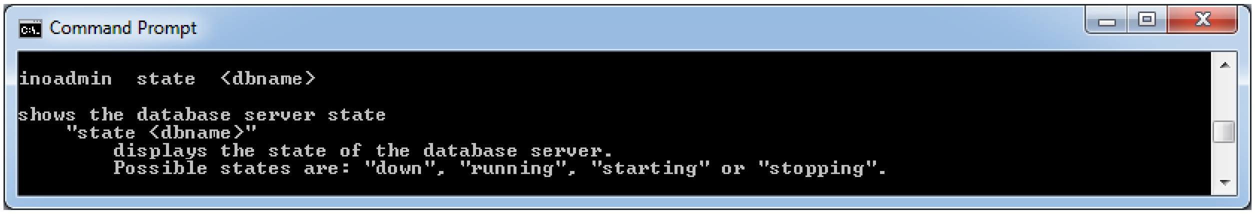
```
inoadmin  stop  <dbname>  { [normal] | rollback | abort }

stops a database server
    "stop <dbname>"  or  "stop <dbname> normal"
        is shutting down the database server
        after waiting for all open transactions to be committed.
    "stop <dbname> rollback"
        is shutting down the database server
        immediately with a rollback of all open transactions.
    "stop <dbname> abort"
        is aborting the database server process.
        A memory dump is written and an autorepair has to be executed with the
        next server start. This option should only be used if a stop in normal
        or rollback mode is not possible.
```

For stopping a database server there are three different options.

The 'normal' option (default) will wait for all open transactions to be committed and thus can take a while. The 'rollback' option is ususally faster but rolls back all open transactions. In both cases, 'normal' as well as 'rollback', the database is afterwards in a consistent state concerning transactions.

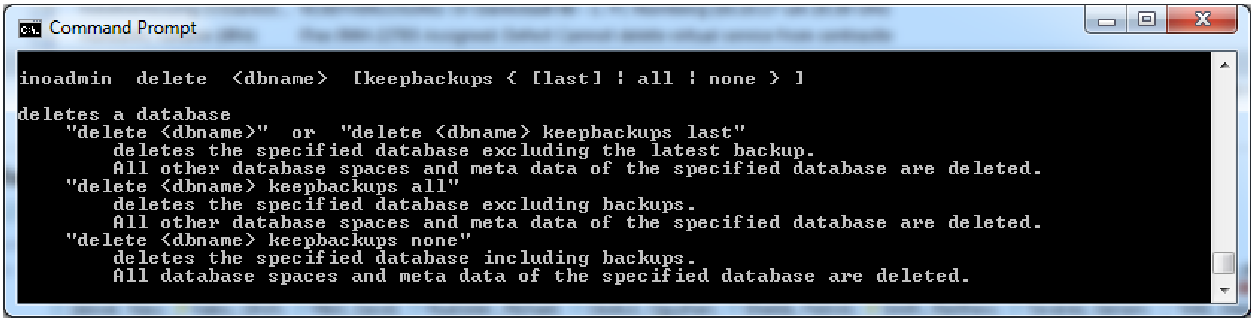
This does not hold for the 'abort' option. This option is only recommended in cases where the database doesn't react to more friendly commands. The database process ends, but a memory dump is written to the server's reserved location. Also, information concerning open transactions gets logged to enable the server to reestablish a consistent state during the next server start's autorepair phase.

The **state** command shows the state of a dabase server identified by its name.



```
inoadmin  state  <dbname>

shows the database server state
    "state <dbname>"
        displays the state of the database server.
        Possible states are: "down", "running", "starting" or "stopping".
```

The **delete** command deletes a dabase identified by its name.

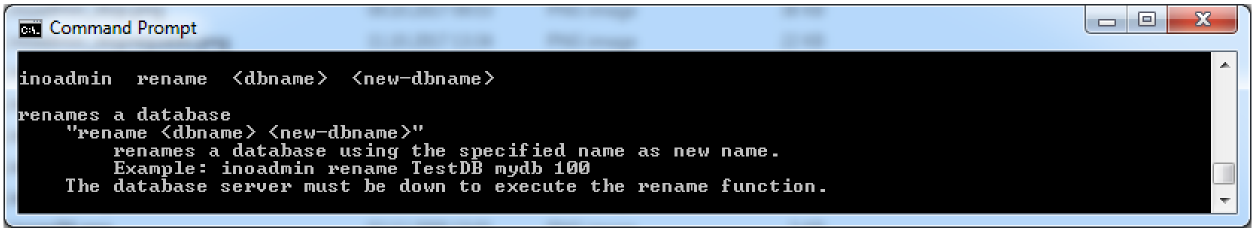


```
inoadmin  delete  <dbname>  [keepbackups { [last] | all | none } ]

deletes a database
    "delete <dbname>"  or  "delete <dbname> keepbackups last"
        deletes the specified database excluding the latest backup.
        All other database spaces and meta data of the specified database are deleted.
    "delete <dbname> keepbackups all"
        deletes the specified database excluding backups.
        All other database spaces and meta data of the specified database are deleted.
    "delete <dbname> keepbackups none"
        deletes the specified database including backups.
        All database spaces and meta data of the specified database are deleted.
```

Deleting a database consists of deleting the data, any server extensions that are installed, and the settings associated with the database. A database can only be deleted if it is inactive (stopped).

Note: Deleting a database does not delete any database locations.

The **rename** command renames a dabase.



```
inoadmin  rename  <dbname>  <new-dbname>

renames a database
    "rename <dbname> <new-dbname>"
        renames a database using the specified name as new name.
        Example: inoadmin rename TestDB mydb 100
    The database server must be down to execute the rename function.
```

Note: When you rename a database, the default group for security (which is usually the same as the database name) is not renamed.

Database Backups

An important part of database administration is to make regular backups of your database. Database Backup and Restore functions are available from the Tamino Manager. The Backup function makes a copy of the database. The Restore function restores the database to the state it was in when you made the backup. Restore is generally executed together with a Recover process, in which all changes that were made to the database after the backup are re-applied. For this process, log files (log spaces) are used as input files. In this way, all data up to the most recently completed transaction can be restored.

The Backup function is used to make a backup copy of a database. You can make Tamino (internal) backups to disk or tape. Backing up the database to an external backup system is also available. Please consult you product support if this is considered.

Caution: If more than one backup is written to a tape, the existing backup will be overwritten. The savepoint of each backup will be offered in the Restore menu, but only the latest backup will be available. No warning will be issued. You should, therefore, not put more than one backup on one tape.

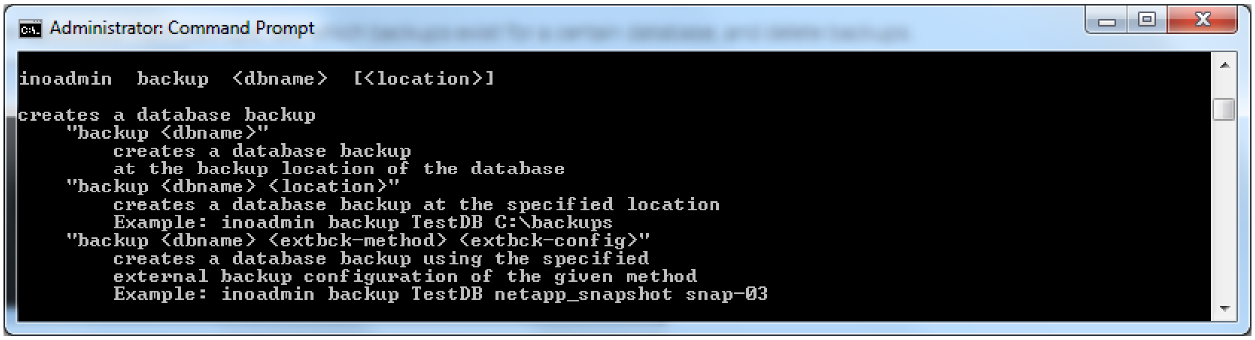
You can make incremental backups of a database. The first incremental backup after a full backup stores the database changes that were made since the full backup. If there are several incremental backups since the most recent full backup, each incremental backup after the first one stores the changes since the preceding incremental backup. Since in general an incremental backup is much smaller than a full backup, the time required to make an incremental backup can be considerably less than the time required to make a full backup.

Incremental backups provide a useful extension to Tamino's backup capabilities. If you decide to include incremental backups in your backup strategy, you should keep a sensible balance between incremental backups and full backups. Regular full backups should still be the central part of your strategy, and the incremental backups can be considered as a quick and safe way of protecting against data loss in the intervals between the full backups.

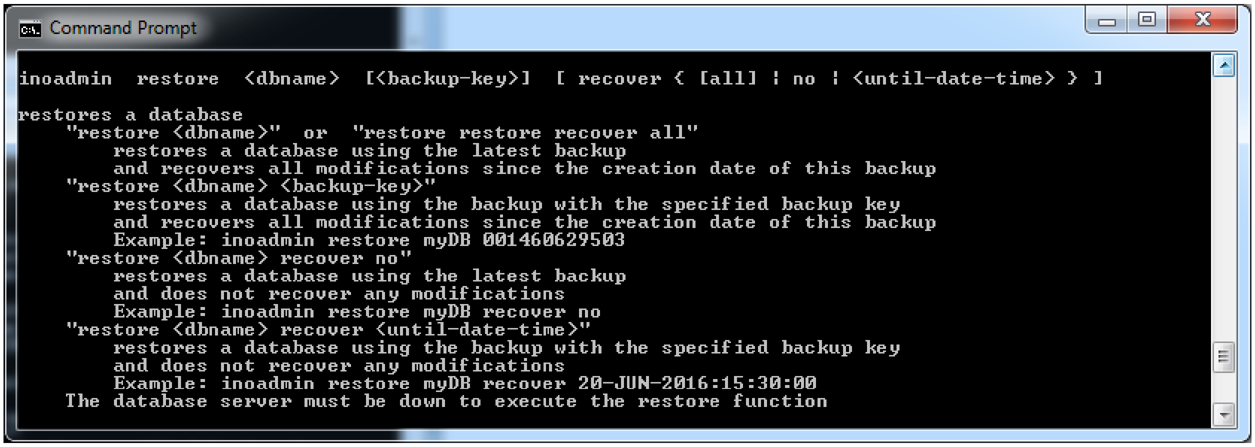
The following general restrictions apply for backups:

- While the Tamino Data Loader utility is running, it is not possible to do a backup.
- Data that has been mapped via Server Extensions will not be backed up, unless it is mapped to the same database (that is being backed up).

The **inoadmin** tool offers commands to create backups, restore databases from backups, see which backups exist for a certain database, and delete backups.



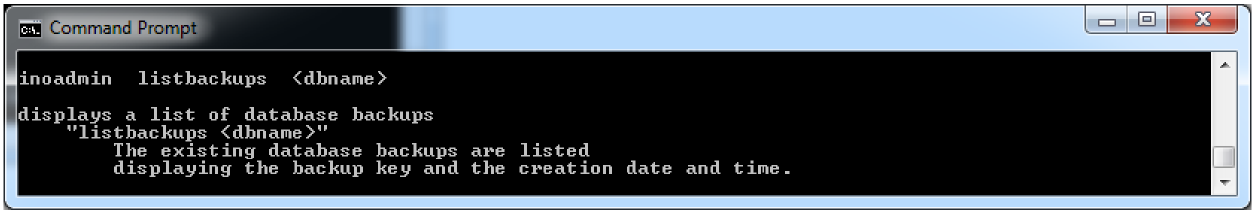
```
inoadmin  backup  <dbname>  [<location>]

creates a database backup
    "backup <dbname>"
        creates a database backup
        at the backup location of the database
    "backup <dbname> <location>"
        creates a database backup at the specified location
        Example: inoadmin backup TestDB C:\backups
    "backup <dbname> <extbck-method> <extbck-config>"
        creates a database backup using the specified
        external backup configuration of the given method
        Example: inoadmin backup TestDB netapp_snapshot snap-03
```



```
inoadmin  restore  <dbname>  [<backup-key>]  [ recover { [all] | no | <until-date-time> } ]

restores a database
    "restore <dbname>"  or  "restore restore recover all"
        restores a database using the latest backup
        and recovers all modifications since the creation date of this backup
    "restore <dbname> <backup-key>"
        restores a database using the backup with the specified backup key
        and recovers all modifications since the creation date of this backup
        Example: inoadmin restore myDB 001460629503
    "restore <dbname> recover no"
        restores a database using the latest backup
        and does not recover any modifications
        Example: inoadmin restore myDB recover no
    "restore <dbname> recover <until-date-time>"
        restores a database using the backup with the specified backup key
        and does not recover any modifications
        Example: inoadmin restore myDB recover 20-JUN-2016:15:30:00
    The database server must be down to execute the restore function
```

Note: During the Restore/Recover step the database is in standby mode, which means that it is not possible to work with it. Hence before and after changing large amounts of data at one point in time, it is recommended to back up the database. Otherwise, the Restore/Recover step might take very long, and the database is not available for that period of time.

Note: If you restore a database from a backup that is not the most recent backup, all of the more recent backups are set to the INVALID state. This means that they cannot be used for subsequent restore operations.

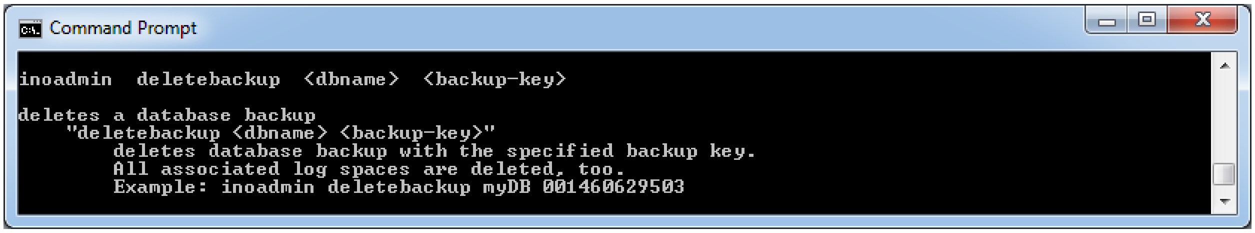


```
inoadmin  listbackups  <dbname>

displays a list of database backups
    "listbackups <dbname>"
        The existing database backups are listed
        displaying the backup key and the creation date and time.
```

In addition to the automatic deletion of backups via Tamino's backup generation feature (defined by the property number of backup generations in the server properties), you can also delete single backups that are no longer required with the **deletebackup** command. Deleting a backup removes all of the backup spaces that are associated with it, but the associated log information is not removed, since it may subsequently be required if the database has to be recovered.

For the deletion of backups the following restrictions apply when using incremental backups:

- If you delete a full backup, Tamino automatically also deletes all subsequent incremental backups that are based on this full backup.
- You cannot select an incremental backup for deletion.



```
inoadmin  deletebackup  <dbname>  <backup-key>

deletes a database backup
    "deletebackup <dbname> <backup-key>"
        deletes database backup with the specified backup key.
        All associated log spaces are deleted, too.
        Example: inoadmin deletebackup myDB 001460629503
```

For further information about backup and restore policies, please refer to the Backup Guide.

Database Properties

You can use **inoadmin** to display information about and modify database properties. These properties are divided into 4 groups:

Server properties
XML properties
Port properties
X-Tension

The following sections provide information about the properties in each group and give instructions on how to display and modify.

Note: The word “yes” in the column “Dynamic” in the following tables indicates that changes to the property value become effective immediately, even if the Tamino server is active. The word “no” indicates a non-dynamic property, which means that the changes only become effective if you stop and restart the database.

Server Properties

Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
auto expand	yes	yes, no	yes	This switch controls whether the Tamino server is allowed to allocate or extend database spaces (data, journal or index) on its own.	■ Tamino first tries to expand an existing space. If this fails, new space will be allocated in the defined reserved location ■ If no free space is available, an error will be returned, depending on the operation performed
autostart	no	yes, no	no	If this switch is set to "yes", the database will be started automatically when the computer starts as well as after its creation.	
buffer pool size	no	5 - 65536	Computed on the basis of the database profile	This is the size of the buffer pool of the Tamino server in MB. The maximum value that you can specify for this property is 65536 MB). However, in practice, the usable buffer pool size reduces to a lower amount, depending on the machine configuration and the Tamino version used.	■ Recommendation: as big as the biggest index ■ Should not be bigger than the physical memory in order to avoid swapping
flush limit	yes	0 - 45875	0	The size in MB of the modified buffer pool space that triggers a flush (disk write) of the modifications present in the buffer pool.	Minimum for flush limit is 1 MB. Setting the flush limit to 0 is also allowed: If this value is set to 0, Tamino will adjust the flush limit itself. If a value higher than 70 % of the buffer pool size is configured, the server will

Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
					internally recalculate the flush limit to 70 % of the buffer pool size, and a warning will be issued during each server startup.
dynamic pool size	no	500 - 61440	500	This is the size in KB of the Tamino server's internal dynamic working pool.	■ Keeps result sets ■ Support for sorting ■ If this value is too small, swapping will occur which leads to increased I/O
maximum transaction duration	yes	20 - 2592000	300	This is the maximum transaction duration in seconds.	This time includes the time for all command processing, including query. Tamino will terminate and roll back transactions that last longer.
non-activity timeout	yes	20 - 2592000	900	The maximum timespan (in seconds) that a user is allowed to be inactive before the Tamino server rolls back pending user transactions.	The non-activity timeout is also used as a communications timeout to the Tamino client. For this purpose, the timeout value used is never greater than 600 seconds.
replication delay	yes	1 - 65535	5	The interval (in seconds) at which the replication databases defined for a master database are updated. The property is set for the master database and applies to all replication databases that are defined for that master database.	Decreasing the value of this property increases the processing overheads associated with more frequent calls for log space data by the replication databases.
flush group size	yes	0 - 5000	0	This property specifies the number of I/Os issued in parallel in a buffer flush. If 0 is specified, the system default is used.	
number of backup generations	yes	1 - (2^{32} -1)	5	The number of full backups that Tamino will keep in parallel. When this number is exceeded, the oldest backup and the corresponding log spaces will be deleted.	

Name	**Dynamic**	**Allowed Values**	**Default Value**	**Description**	**Additional Information**
				Backup generations refer to full backups only, not incremental backups.	
read only	no	yes, no	no	The database is read only. No insert, update or delete operations are possible.	Database cannot be deleted when this property is set to "yes".
tolerate autorepair errors	no	yes, no	yes	If this is set to yes, errors that occur during the autorepair of the Tamino server after an abnormal termination will be tolerated rather than leading to a refusal to start. The indexes or doctypes concerned will be disabled.	A disabled index can be fixed using the Check and Repair command in the Tamino Manager
server host name	no		none	This defines the host name part for the Tamino Server's registration with XTS.	■ The Tamino server cannot be contacted from other nodes via XTS. Instead, applications on other nodes will look for the server on their node. ■ This setting can be used on portable devices to ensure that an interruption of network connections does not cause any problems with administrating and communicating with the Tamino server. ■ Max. length=255
communication method	no	SSL and TCP/IP, SSL, XTS and TCP/IP, XTS, TCP/IP, HTTP, HTTPS	XTS and TCP/IP	This defines which standard method Tamino uses for communication.	If TCP/IP is used in combination with XTS or SSL, XML requests can be sent to the server via XTS and TCP/IP. (Note that using SSL requires a special certificate) If XTS or SSL are used, the XML port is not allocated and XML requests can only be sent via XTS.

Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
					If TCP/IP is used, the XML XTS port is not allocated and XML requests can only be sent via TCP/IP. If HTTP or HTTPS is used, property HTTP port must be set.
SSL certificate file	no	max. length 1023	none	Specifies the file name of the server's certificate.	
SSL key file	no	max. length 1023	none	Specifies the file name of the server's key.	Optional. The key can also be included in the certificate file.
SSL password	no	max. length 1023	none	Holds the pass phrase of the server's SSL key.	Optional. Used to decrypt the file specified in SSL key file, if this file is encrypted.
SSL CA file	no	max. length 1023	none	Specifes the file name of the trust file	To be used for certificate base authentication and encryption
SSL certificate chain verification depth	no	1023	1	Specifies the maximum depth of chained certificate evaluation	
SSL verify_client	no	yes, no	no	Indicates whether the client is to be authenticated by the Tamino server via SSL	
deprecated features	yes	tolerate, log, log and reject	log	Specifies how Tamino reacts to the usage of deprecated features. It can react by executing the request and writing an error to the session's job log (`log`), by additionally rejecting the request which uses the deprecated feature (`log and reject`), or by silently executing the request (`tolerate`).	
unbuffered	no	none, all, database, journal	none	This property specifies which parts of a Tamino database are handled with unbuffered I/O. The property usually increases I/O times but	value `database`: means data spaces and index spaces in this context

Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
				removes the need for synchronization on devices.	
overwrite XTS registration	no	yes, no	no	This switch controls whether the Tamino server should remove existing XTS directory service entries for the server and create new entries during startup.	Before setting this value to “yes”, check that no database with the same name is active on another node, using the same XTS directory server. You would disable any client communication with such a database.
incremental backup	yes	yes, no	no	This switch controls whether an incremental backup can be made for the database.	

XML Properties

Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
XQuery document cache size	yes		20	Specifies the capacity of the document cache that is used for XQuery processing.	Document caching improves the performance of certain queries like join queries where the sorting is done via an index. Also certain XQuery update statements can benefit from a big document cache. The capacity of the XQuery document cache is specified in MB.
word fragment index	no	yes, no	no	If this is set to yes, a word fragment index is maintained.	Maintaining a word fragment index can be very time- and resource-intensive, but will increase performance for certain query types. For read-only databases (e.g. replication

Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
					databases), the word fragment index parameter is displayed, but cannot be modified.
XML maximum sessions	no	0 - 100000	100	The maximum number of XML sessions that can be open at the same time.	A sessionless request counts as a session for the execution time.
XML max simultaneous requests	no	150 - 10000	0	Specifies the size of the queue of free XML request entries and therefore limits the maximum numbers of simultaneous requests that can be accepted by the Tamino Server.	The default value (0) means that the actual value of the property is computed by Tamino at startup time. The value for a database with default settings is 157 (100 + XML work threads + XML dynamic work threads + 2).
XML work threads	no	1 - 1024	5	The number of threads used to process XML commands.	Requests are queued if there are no XML work threads available.
X-Node default encoding	no	supported encoding	ISO-8859-1	The encoding that external data sources which are accessed by Tamino use for their data.	Can be overwritten within the schema definition where the mapping is specified.
X-Node default password	no	Character string (max. 128 characters)	guest	The password used for ODBC connections if no other password is specified.	Can be overwritten within the schema definition where the mapping is specified.
X-Node default user	no	Character string (max. 128 characters)	guest	The user-ID used for ODBC connections if no other user ID is specified.	Can be overwritten within the schema definition where the mapping is specified.
default tokenizer	no	white space-separated, Japanese	white space-separated	Specifies the component that is used to break strings into words.	
default encoding	no	Encoding names (see list	ISO-8859-1	Specifies the encoding used for responses from	

Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
		box)specified in the **IANA registry**		Tamino if no explicit encoding is requested.	
XML document default encoding	no	UTF-8 and ISO-8859-1	UTF-8	Specifies the encoding that is assumed for a received document if the encoding is not specified otherwise.	
stemming index	no	yes, no	no	Specifies whether or not a stemming index is maintained. A stemming index speeds up text retrieval operations that use stemming functionality.	For read-only databases, the stemming index parameter is displayed, but cannot be modified.
markup as delimiter	no	yes, no, mixed	no	Specifies how markup is treated. If set to "yes", markup is treated as a delimiter for text (i.e. markup is handled like white space). If set to "mixed", markup is not treated as a delimiter in mixed-content elements.	If you change this property, it is necessary to recreate all text indexes.
authentication	no	web server, tamino, none	web server	Specifies whether authentication is performed by the web server, by Tamino, or not at all.	Please refer to the section **Tamino Authentication** for further information about this property.
internal authentication file	no	Character string (max. 128 characters)		Points to the file containing users and passwords if file-based authentication is utilitzed.	Please refer to the section **Tamino Authentication** for further information about this property.
XML maximum request duration	yes	20 - 2592000	300	The maximum duration of each request is limited to the specified number of seconds.	After this period, the request terminates with an error, but the transaction is not rolled back.
queue next request	yes	yes, no	yes	Specifies whether a subsequent request in a session is allowed even if the result of the previous request has not been completely processed. If	

Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
				set to “yes”, a subsequent request in a session is allowed while the previous request is still active. If set to “no”, a subsequent request in a session is not allowed if the previous request is still active.	
reject xsi:type	no	yes, no	yes	Rejects xsi:type attribute in element validation.	
text index word length	no	32-250	64	Specifies the maximum length for text index entries.	
XML text index type	no	single word, double word	double word	Specifies if a text query may contain one or two words (single-word index or double-word index).	Usage of three or more word queries cannot be an indexed search, thus resulting in low performance.
request log folder	no	-	-	This is the folder/directory where XML request log files are stored.	The default folder for the request log is the reserved location, if no request log folder is specified.
request log	yes	off, full, requests	off	This property specifies whether requests to Tamino are logged to a file. Logging can be restricted to the pure request (requests), or the requests with all data sent to Tamino (full). The default is that no logging is performed (off).	
request log file count	no	0 - 999	2	This is the maximum number of XML request log files written. If 0 is specified, XML request logging is turned off.	
request log file maximum size	no	0 - 2997152	1	This is the maximum size of an XML request log file (in MB). If this size is reached, Tamino switches	

Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
				to a new request log file. If the last log file is full, Tamino switches to the first one.	

Port Properties

Port numbers should only be changed if there is a special reason to do so. Otherwise, it is recommended to take the default, as Tamino handles port allocation dynamically.

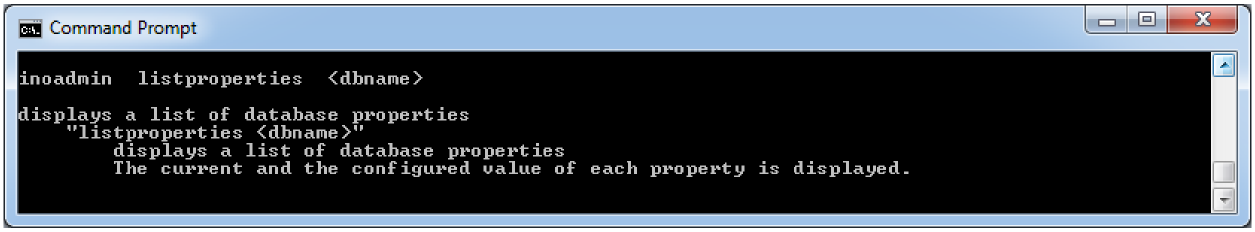
Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
replication port	no	0 or 1024 - 65535	0 (dynamic allocation)	The number of the port used for access to the Tamino server for replication databases	Only used by replication databases. A value of 0 means that the port number is allocated dynamically at database startup. If you do not use the value 0, the number ■ should be taken from the range of static port numbers defined during installation ■ should be changed if there are conflicts with other applications
XML port	no	0 or 1024 - 65535	default from static port range	The number of the port used for XML access to the Tamino server via pure TCP/IP	A value of 0 means that the port number is allocated automatically. If you do not use the value 0, the number ■ should be taken from the range of static port numbers defined during installation ■ should be changed if there are conflicts with other applications
administration port	no	0 or 1024 - 65535	0 (dynamic allocation)	The number of the port used for access to the Tamino server for all administration actions	A value of 0 means that the port number is allocated dynamically at database startup. If you do not use the value 0, the number ■ should be taken from the range of static port numbers defined during installation ■ should be changed if there are conflicts with other applications

Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
XML XTS port	no	0 or 1024 - 65535	0 (dynamic allocation)	The number of the port used for XML access to the Tamino server when using Software AG's extended transport service for XML access, including encrypted communication via SSL.	A value of 0 means that the port number is allocated dynamically at database startup. If you do not use the value 0, the number ■ should be taken from the range of static port numbers defined during installation ■ should be changed if there are conflicts with other applications
HTTP port	no	0 or 1024 - 65535	default from static port range	The number of the port used for XML access to the Tamino server when using native HTTP or HTTPS communication.	A value of 0 means that the port number is allocated automatically. If you do not use the value 0, the number ■ • should be taken from the range of static port numbers defined during installation ■ • should be changed if there are conflicts with other applications

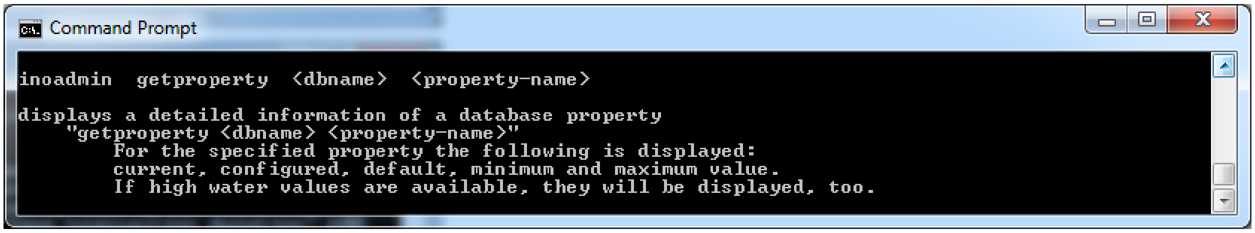
X-Tension Properties

Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
X-Tension Java Options	no	Character string (max. 255 characters)	none	This is used to specify optional properties for the JVM that is started when X-Tension Java usage is turned on.	The syntax of the string is as follows: The first character of the string serves as the delimiter for the options that follow. For example: ?opt1?opt2?opt3 results in the options opt1, opt2 and opt3.
X-Tension Java usage	no	yes, no	yes	If this switch is set to yes, the X-Tension Java infrastructure is enabled. If this switch is set to no, Java-based X-Tensions cannot be installed, modified, upgraded or executed, even if they were installed before the switch was set to no.	

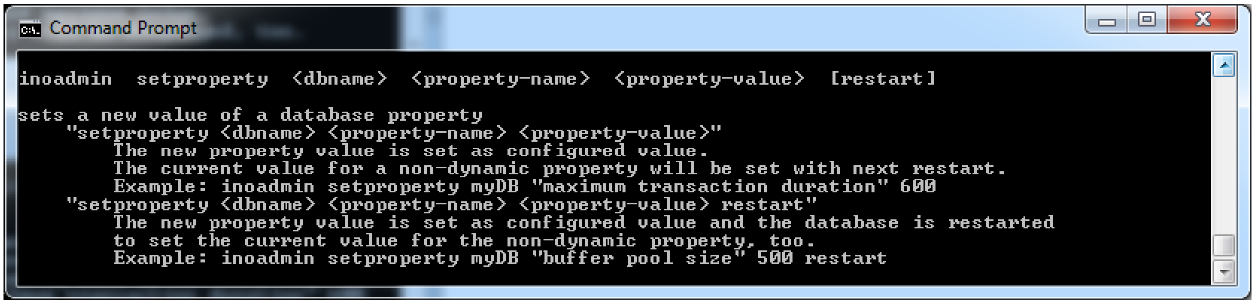
Name	Dynamic	Allowed Values	Default Value	Description	Additional Information
X-Tension COM usage	no	yes, no	yes	If this switch is set to yes, the X-Tension COM infrastructure is enabled. If this switch is set to no, COM X-Tensions cannot be installed, modified, upgraded or executed, even if they were installed before the switch was set to no.	
X-Tension direct usage	no	yes, no	yes	If this switch is set to yes, the X-Tension Direct infrastructure is enabled. If this switch is set to no, Direct X-Tensions cannot be installed, modified, upgraded or executed, even if they were installed before the switch was set to no.	



```
inoadmin  listproperties  <dbname>

displays a list of database properties
    "listproperties <dbname>"
        displays a list of database properties
        The current and the configured value of each property is displayed.
```



```
inoadmin  getproperty  <dbname>  <property-name>

displays a detailed information of a database property
    "getproperty <dbname> <property-name>"
        For the specified property the following is displayed:
        current, configured, default, minimum and maximum value.
        If high water values are available, they will be displayed, too.
```



```
inoadmin  setproperty  <dbname>  <property-name>  <property-value>  [restart]

sets a new value of a database property
    "setproperty <dbname> <property-name> <property-value>"
        The new property value is set as configured value.
        The current value for a non-dynamic property will be set with next restart.
        Example: inoadmin setproperty myDB "maximum transaction duration" 600
    "setproperty <dbname> <property-name> <property-value> restart"
        The new property value is set as configured value and the database is restarted
        to set the current value for the non-dynamic property, too.
        Example: inoadmin setproperty myDB "buffer pool size" 500 restart
```

Replication

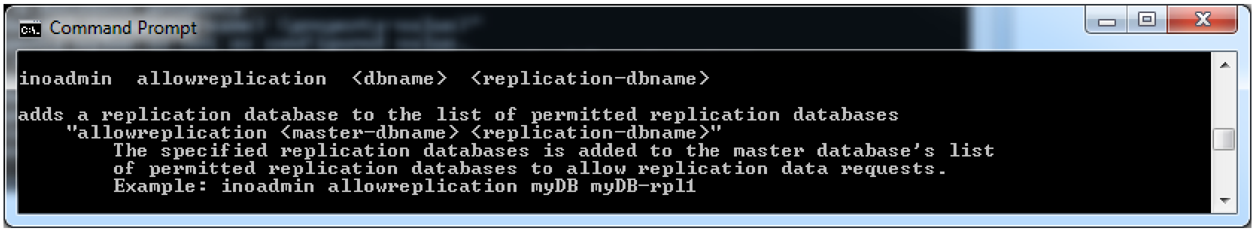
You can use **inoadmin** to create and manage replication databases.

The first step in creating a replication database is to create an exact copy (slave) of the database to be replicated (master) from the most recent backup of that database. To do so, you first have to create a database from a (non-initial) backup of your current database. For further information about creating a database from a backup, please refer to the section **Database Backups**.

Caution: Do not start a replication database before you have performed "inoadmin setreplication".

Once the database has been created from backup, its name has to be added to the list of permitted replication databases for the master database. A master database can have one or more replication databases. The database to which the replication database name is to be added must be active (running).

The **allowreplication** command adds a replication database to a master database.

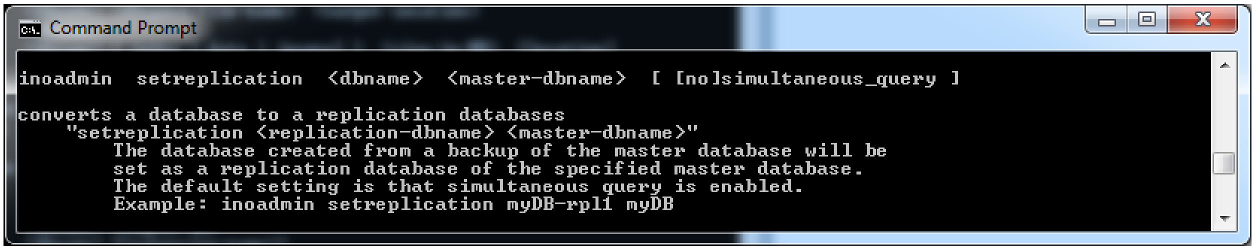


```
inoadmin  allowreplication  <dbname>  <replication-dbname>

adds a replication database to the list of permitted replication databases
    "allowreplication <master-dbname> <replication-dbname>"
        The specified replication databases is added to the master database's list
        of permitted replication databases to allow replication data requests.
        Example: inoadmin allowreplication myDB myDB-rpl1
```

It is possible to enter the name of a replication database that has not yet been created from backup, but will be in the future. Thus, you can create a “pool” of replication database names to be used at a later point in time, so that you do not have to repeat this step each time a new replication database is to be created.

Note: The first time you add an entry to the list it is not necessary to restart the master database.

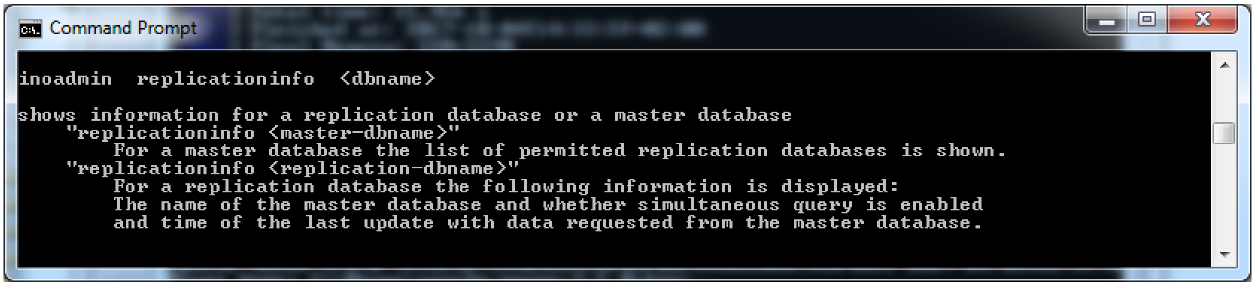
Once the name of the replication database has been added to the master database, the replication database must be set as a replication database. This is done with the **setreplication** command.



```
inoadmin  setreplication  <dbname>  <master-dbname>  [ [no]simultaneous_query ]

converts a database to a replication databases
    "setreplication <replication-dbname> <master-dbname>"
        The database created from a backup of the master database will be
        set as a replication database of the specified master database.
        The default setting is that simultaneous query is enabled.
        Example: inoadmin setreplication myDB-rpl1 myDB
```

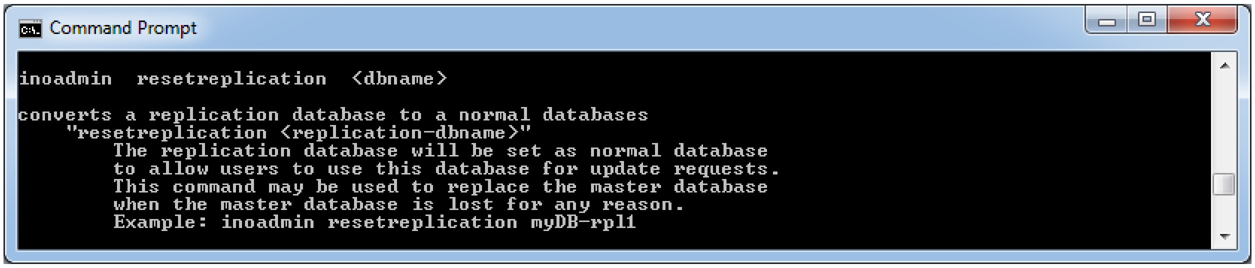
The replication database must be inactive (stopped).

Further commands are provided to obtain information about the current status of a database w.r.t. replication, to switch a replication database back again to a non-replication database, and to drop a database from a list of a master's possible replication databases.



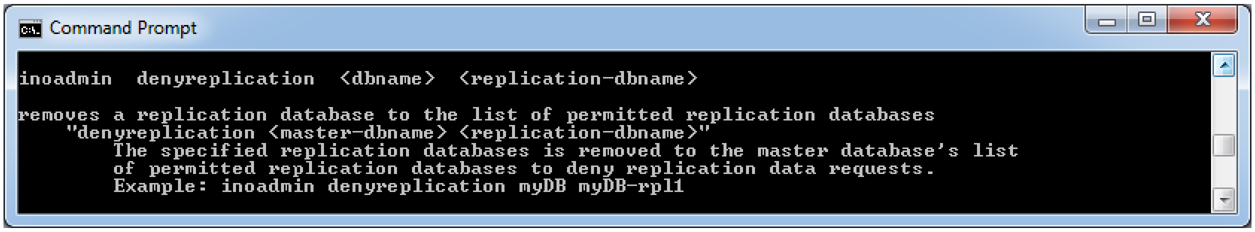
```
inoadmin  replicationinfo  <dbname>

shows information for a replication database or a master database
    "replicationinfo <master-dbname>"
        For a master database the list of permitted replication databases is shown.
    "replicationinfo <replication-dbname>"
        For a replication database the following information is displayed:
        The name of the master database and whether simultaneous query is enabled
        and time of the last update with data requested from the master database.
```

If, for any reason, the master database is lost, you can reset the replication database to a normal database, which can then be used as a substitute for the lost master. The replication database to be reset must be inactive (stopped).



```
inoadmin  resetreplication  <dbname>

converts a replication database to a normal databases
    "resetreplication <replication-dbname>"
        The replication database will be set as normal database
        to allow users to use this database for update requests.
        This command may be used to replace the master database
        when the master database is lost for any reason.
        Example: inoadmin resetreplication myDB-rpl1
```

Note: Once you have reset a replication database, replication cannot be continued.

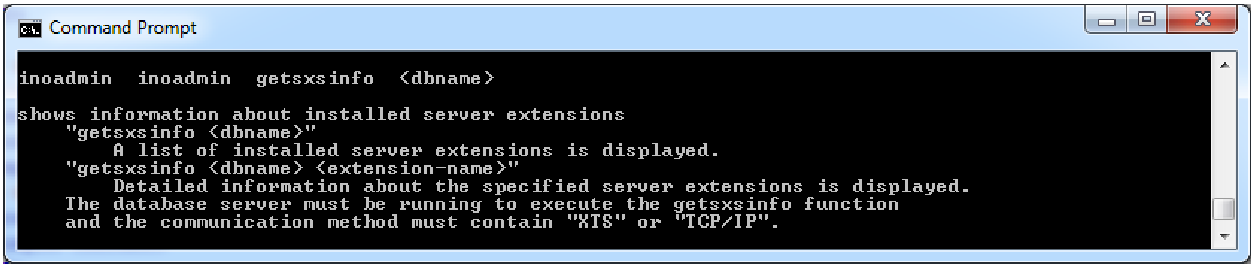


```
inoadmin  denyreplication  <dbname>  <replication-dbname>

removes a replication database to the list of permitted replication databases
    "denyreplication <master-dbname> <replication-dbname>"
        The specified replication databases is removed to the master database's list
        of permitted replication databases to deny replication data requests.
        Example: inoadmin denyreplication myDB myDB-rpl1
```

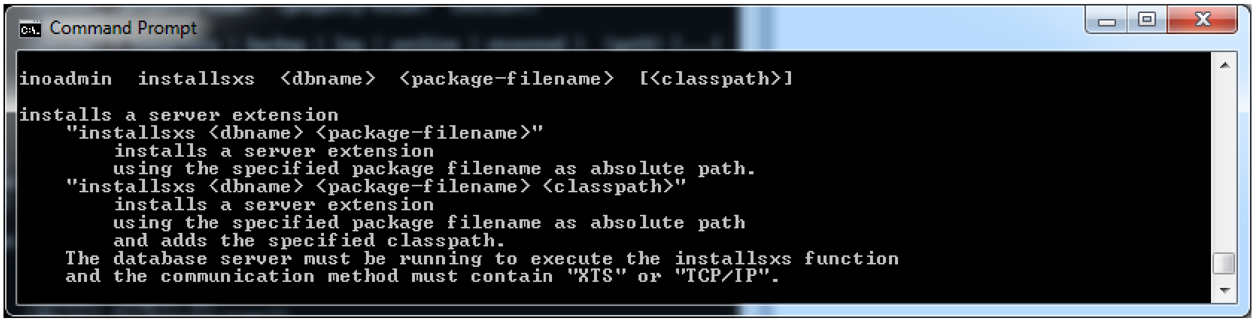
For further information about database replication and when to use it, please refer to the section *Replication Guide*.

Tamino Server Extensions (SXS)

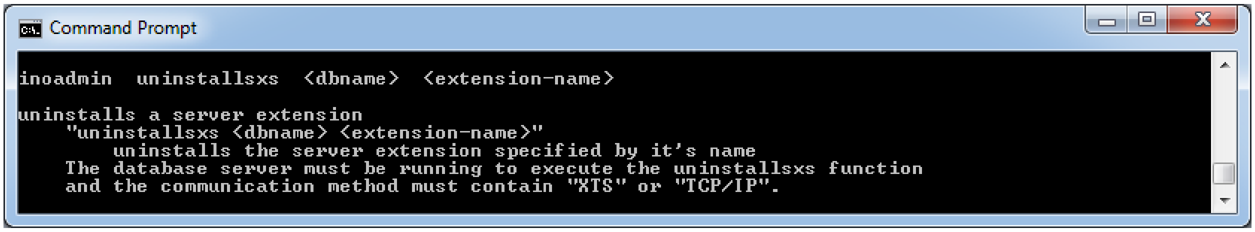
You can use **inoadmin** to administer Tamino Server Extensions. Commands are provided for obtaining information upon currently installed server extensions and installing and uninstalling extensions.



```
inoadmin  inoadmin  getsxsinfo  <dbname>

shows information about installed server extensions
    "getsxsinfo <dbname>"
        A list of installed server extensions is displayed.
    "getsxsinfo <dbname> <extension-name>"
        Detailed information about the specified server extensions is displayed.
    The database server must be running to execute the getsxsinfo function
    and the communication method must contain "XTS" or "TCP/IP".
```



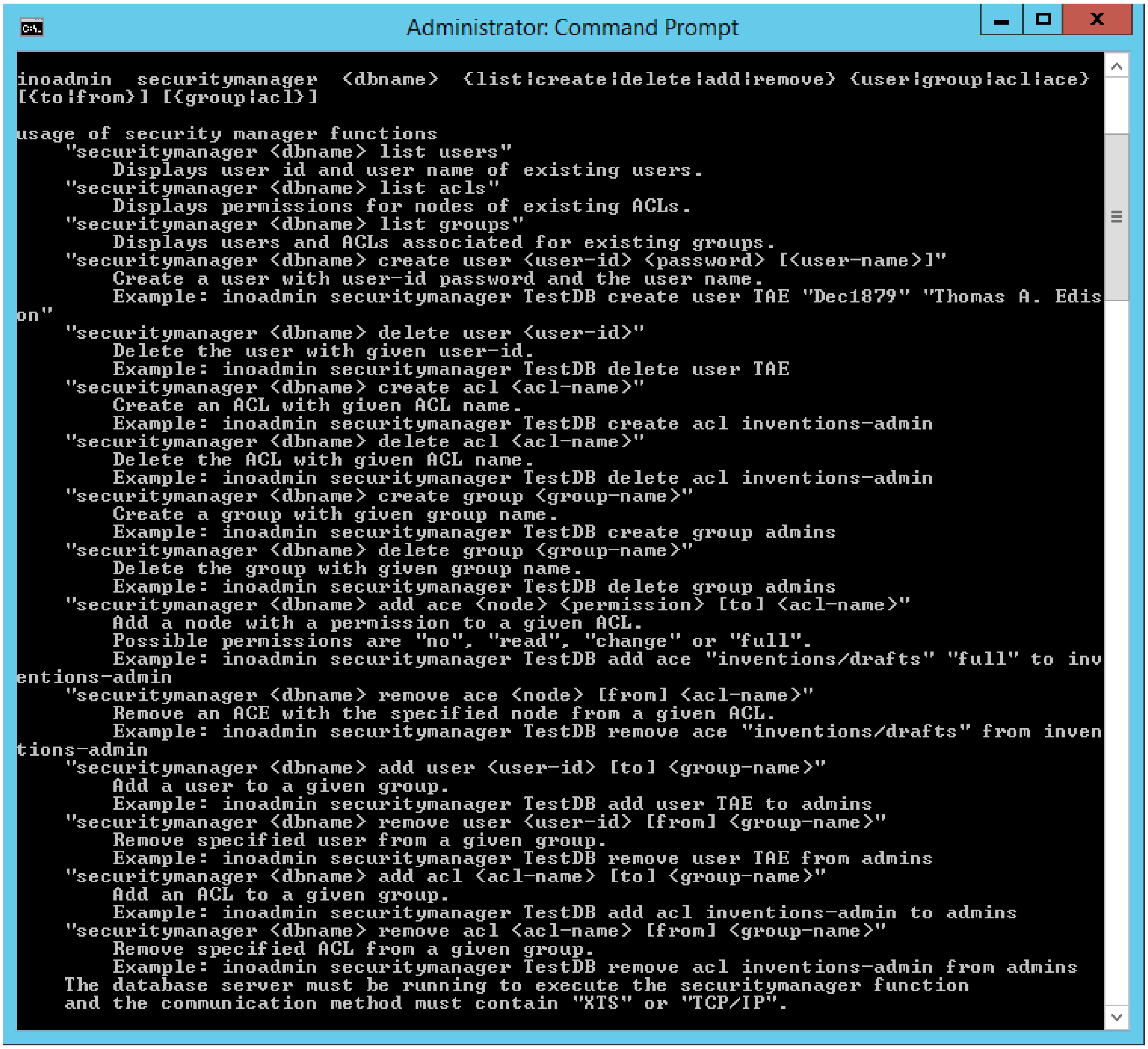
```
inoadmin  installsxs  <dbname>  <package-filename>  [<classpath>]

installs a server extension
    "installsxs <dbname> <package-filename>"
        installs a server extension
        using the specified package filename as absolute path.
    "installsxs <dbname> <package-filename> <classpath>"
        installs a server extension
        using the specified package filename as absolute path
        and adds the specified classpath.
    The database server must be running to execute the installsxs function
    and the communication method must contain "XTS" or "TCP/IP".
```



```
inoadmin  uninstallsxs  <dbname>  <extension-name>

uninstalls a server extension
    "uninstallsxs <dbname> <extension-name>"
        uninstalls the server extension specified by it's name
    The database server must be running to execute the uninstallsxs function
    and the communication method must contain "XTS" or "TCP/IP".
```

For further information about server extensions and how to use them, please refer to the section *X-Tension: Tamino Server Extensions*.

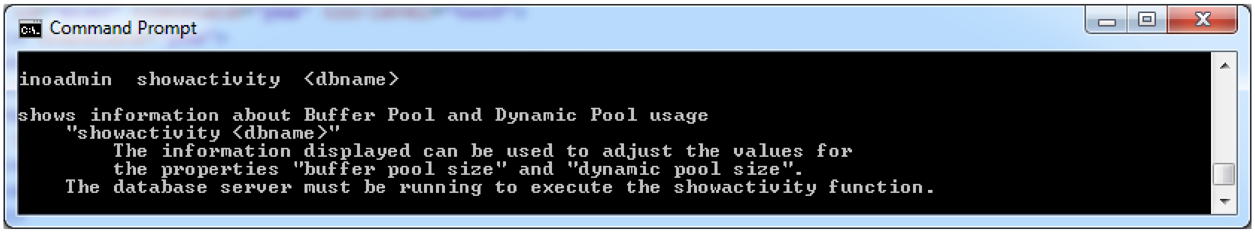
inoadmin Security Manager



```
inoadmin  securitymanager  <dbname>  {list|create|delete|add|remove} {user|group|acl|ace}
[{to|from}] [{group|acl}]

usage of security manager functions
    "securitymanager <dbname> list users"
        Displays user id and user name of existing users.
    "securitymanager <dbname> list acls"
        Displays permissions for nodes of existing ACLs.
    "securitymanager <dbname> list groups"
        Displays users and ACLs associated for existing groups.
    "securitymanager <dbname> create user <user-id> <password> [<user-name>]"
        Create a user with user-id password and the user name.
        Example: inoadmin securitymanager TestDB create user TAE "Dec1879" "Thomas A. Edis
on"
    "securitymanager <dbname> delete user <user-id>"
        Delete the user with given user-id.
        Example: inoadmin securitymanager TestDB delete user TAE
    "securitymanager <dbname> create acl <acl-name>"
        Create an ACL with given ACL name.
        Example: inoadmin securitymanager TestDB create acl inventions-admin
    "securitymanager <dbname> delete acl <acl-name>"
        Delete the ACL with given ACL name.
        Example: inoadmin securitymanager TestDB delete acl inventions-admin
    "securitymanager <dbname> create group <group-name>"
        Create a group with given group name.
        Example: inoadmin securitymanager TestDB create group admins
    "securitymanager <dbname> delete group <group-name>"
        Delete the group with given group name.
        Example: inoadmin securitymanager TestDB delete group admins
    "securitymanager <dbname> add ace <node> <permission> [to] <acl-name>"
        Add a node with a permission to a given ACL.
        Possible permissions are "no", "read", "change" or "full".
        Example: inoadmin securitymanager TestDB add ace "inventions/drafts" "full" to inv
entions-admin
    "securitymanager <dbname> remove ace <node> [from] <acl-name>"
        Remove an ACE with the specified node from a given ACL.
        Example: inoadmin securitymanager TestDB remove ace "inventions/drafts" from inven
tions-admin
    "securitymanager <dbname> add user <user-id> [to] <group-name>"
        Add a user to a given group.
        Example: inoadmin securitymanager TestDB add user TAE to admins
    "securitymanager <dbname> remove user <user-id> [from] <group-name>"
        Remove specified user from a given group.
        Example: inoadmin securitymanager TestDB remove user TAE from admins
    "securitymanager <dbname> add acl <acl-name> [to] <group-name>"
        Add an ACL to a given group.
        Example: inoadmin securitymanager TestDB add acl inventions-admin to admins
    "securitymanager <dbname> remove acl <acl-name> [from] <group-name>"
        Remove specified ACL from a given group.
        Example: inoadmin securitymanager TestDB remove acl inventions-admin from admins
    The database server must be running to execute the securitymanager function
    and the communication method must contain "XTS" or "TCP/IP".
```

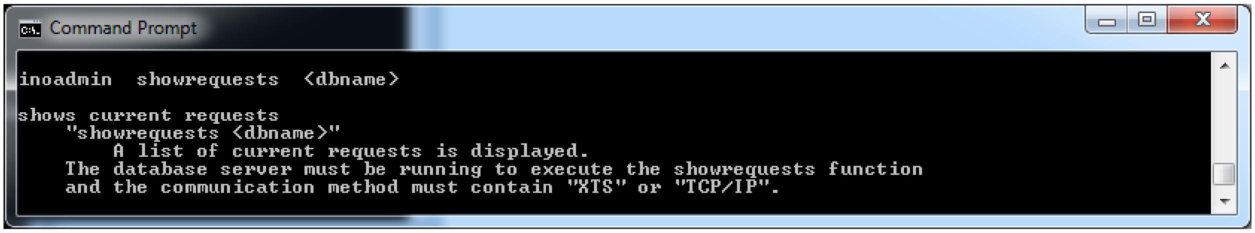
The **securitymanager** command allows to create and maintain the objects that rule the Tamino security settings like groups, users, and ACLs. For further information about Tamino security, please refer to the section *Tamino Security*.

Miscellaneous

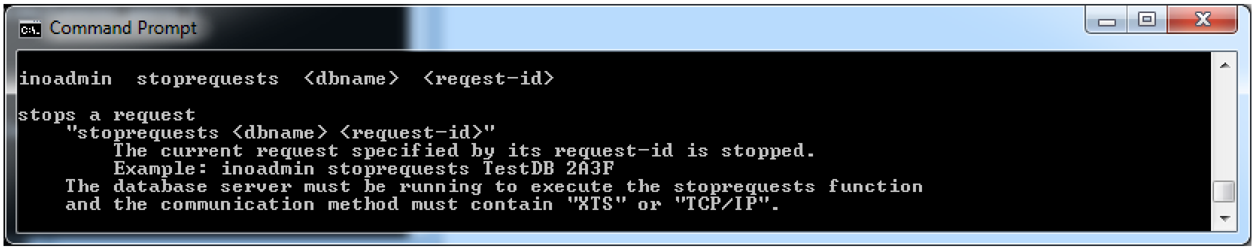


```
inoadmin  showactivity  <dbname>

shows information about Buffer Pool and Dynamic Pool usage
    "showactivity <dbname>"
        The information displayed can be used to adjust the values for
        the properties "buffer pool size" and "dynamic pool size".
    The database server must be running to execute the showactivity function.
```

The commands concerning database requests allow to show the requests that are currently running against Tamino, and to stop a specific requests.

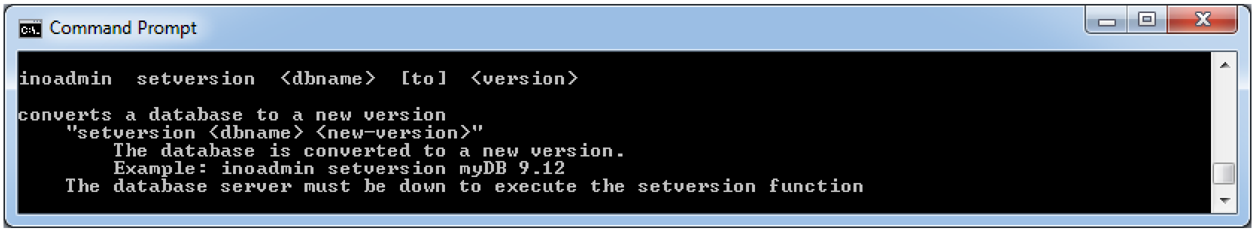


```
inoadmin  showrequests  <dbname>

shows current requests
    "showrequests <dbname>"
        A list of current requests is displayed.
    The database server must be running to execute the showrequests function
    and the communication method must contain "XTS" or "TCP/IP".
```



```
inoadmin  stoprequests  <dbname>  <reqest-id>

stops a request
    "stoprequests <dbname> <request-id>"
        The current request specified by its request-id is stopped.
        Example: inoadmin stoprequests TestDB 2A3F
    The database server must be running to execute the stoprequests function
    and the communication method must contain "XTS" or "TCP/IP".
```

You can use the **setversion** command to change the version of a database when you upgrade/downgrade from one version of Tamino to another. This means that databases created with earlier versions of Tamino can still be used with subsequent versions.



```
inoadmin  setversion  <dbname>  [to]  <version>

converts a database to a new version
    "setversion <dbname> <new-version>"
        The database is converted to a new version.
        Example: inoadmin setversion myDB 9.12
    The database server must be down to execute the setversion function
```

It is highly recommended only to use the **setversion** command when the database is stopped and no other administration actions are in progress (for example, recovery). The database should generally be started immediately after **setversion**, so that standard automatic re-initialization steps can be carried out. It is also recommended to back up a database after a successful downgrade, otherwise a complete restore/repair will no longer be possible.

Note: The authentication mode of the database needs to be set to "none" before a database can be upgraded via `setversion`. For more information see section *Authentication Modes*.

4 Tamino Security

This chapter contains information about Tamino authentication and the Tamino authorization concept. The following topics are covered:

Tamino Authentication

Tamino supports an authentication mechanism that uses an external web server or information that is stored in the database. When a web server is used to perform authentication, other mechanisms (in addition to the HTTP basic authentication mechanism) that are supported by that web server can also be used. When Tamino is used to perform the authentication, only the basic authentication mechanism is supported.

Authentication Modes

There are 3 possible authentication modes. The authentication mode is defined per database with the XML property `Authentication`. Please refer to **Database Properties** for further information about how to set and modify the authentication mode.

The following authentication modes are available:

web server

In this mode, Tamino receives the userid from the web server module and relies on the web server to authenticate the user. If this mode is set and the web server is not configured to perform authentication, a blank userid will be used in Tamino which will be assigned to the permissions of the default group. If a request is authenticated successfully and passed to Tamino, the userid (and the domain) specified in the HTTP basic authentication scheme is used.

This is the default mode that is set when a new database is created. If this method is used, the web servers that are allowed to communicate with Tamino should be carefully restricted.

tamino

This mode specifies that authentication is performed directly by Tamino. All requests sent to Tamino are authenticated against the users known to Tamino. These users are either stored directly in Tamino in the collection *ino:security* or are stored in an external authentication service which Tamino can query (see section **External Authentication** in this chapter). Each request sent to Tamino must include a userid and a password; if these are not provided or if authentication fails, Tamino will return an HTTP response code 401.

none

In this mode, Tamino will use the userid provided for authorization, but it will not be authenticated. The userid can be sent in either the HTTP basic authentication header field or in the special header field `X-INO-Authorization`.

Note: There is an additional mode “tamino no challenge”. This mode is equivalent to “tamino”, but provides some additional internal functionality which is used by certain

Software AG products. If your installation of Tamino is preset to "tamino no challenge", do not change that mode.

Passwords

If a user is to be authenticated by Tamino, there must be a password stored inside Tamino for that user. Passwords are not stored as plain text, but rather as hash values that are computed using a one-way hash function. In order to store a password, the plaintext password has to be put into the attribute `ino:unencryptedpassword`. The value in this attribute is then hashed, and the resulting value is stored in the attribute `ino:password`. If the password was retrieved as a hash value, it can be stored back into Tamino directly using the attribute `ino:password`.

A user can change his own password with the special administration function `ino:ChangeUserPassword`. This function changes the password of the user that issues the command, and will only work when the authentication mode is set to `Tamino`. The function has only one parameter, which is the new password in plaintext:

Example

```
_admin=ino:ChangeUserPassword("new_password")
```

The current request will be executed using the old password, but the next request for the user in question will have to be executed using the new password. Executing this function also sets the attribute `ino:passwordlastchanged` to the current system time.

Note: In order to avoid plaintext passwords from being stored in web server access logs, this admin command should always be sent using an HTTP POST request.

External Authentication

When using the authentication mode `tamino`, it is possible to have Tamino check the user credentials against a local operating system, an internal text file, or against an LDAP Server. In order to use this feature, users must be grouped into domains, where each domain is associated with an authentication service.

Domains are defined in the doctype *ino:domain* of the collection *ino:security*. A domain defines a unique name and the type of authentication service and possibly some additional parameters to describe the authentication service. The following example shows the definition of a domain *`mydomain`*, which uses the local operating system to authenticate the users:

```
<ino:domain ino:domainid="mydomain" ino:domaintype="os"
xmlns:ino="http://namespaces.softwareag.com/tamino/response2" />
```

The following attributes can be set for a domain:

ino:domainid (required)
Specifies the unique name of the domain - this unique name must be specified as part of the user names in order to uniquely assign a user to a domain.

ino:domaintype (optional - default=tamino)
Specifies the type of authentication service. Possible values are:

- tamino: authenticate against users whose passwords are defined in Tamino
- os: authenticate against the local operating system
- ldap: authenticate against an external LDAP server
- internal: authenticate against an internal text file

ino:expire (optional - default = 0)
The number of seconds that a user should be cached in Tamino before being reauthenticated against the external service. A value of 0 means that the user is reauthenticated with every request sent to Tamino. When using a local operating system, this value should be set fairly low and when using an external service which is connected via a network, this value should be set higher, in order to avoid unnecessary network accesses.

ino:acceptusers (optional - default = defined)
This option specifies whether to allow access of any user that is correctly authenticated by the authentication service (value `all`) or whether to only allow access by users that are explicitly defined in the `ino:user` doctype (value `defined`).

ino:usegroups (optional - default = false)
This option specifies whether to use the external group information from domains – for example the groups in an Active Directory Server or in an LDAP server. When this attribute is set to true, the external groups will be visible and will be preceded by the domain name and a backslash. Note that this option can only be set by directly storing the XML configuration files into Tamino.

ino:default (optional - default = false)
This option specifies whether the domain should be used as default authentication domain. Setting this option will result in users being authenticated against this domain, even when no domain name is given with the userid. If a user exists which is defined locally in Tamino and also in the default domain, the local user will be used for authentication.

Only one domain can be defined of the domain type *`internal`*, and its name needs to be `INTERNAL`. The file that is used for authentication of the domain's users is the one pointed to from the XML property `internal authentication file` which is initialy empty.

A Tamino installation comes with a pre-configured authentication file *<INSTALL_ROOT>/common/conf/users.txt* that already contains Administrator/manage. Thus setting the property `internal` `authentication file` accordingly should allow logging in with the user INTERNAL\Administrator using the password 'manage'.

The internal authentication file can be modified with the tool *<INSTALL_ROOT>/common/bin/internaluserrepo.bat*. For help enter **internaluserrepo.bat -help**.

When specifying the domain type `ldap`, some additional parameters need to be set, to describe the location and configuration of the LDAP server. For this purpose, key-value-pairs can be specified as contents of the ino:domain element in the following format:

```
<ino:param ino:name=".." ino:content=".." />
```

The following parameters need to be specified for an LDAP:

`host`	IP number of the host where the LDAP server is running
`port`	Port on which the LDAP server is listening
`ldap_person_dn`	The distinguished name of the person entries
`ldap_person_object`	The class hierarchy for person entries from the top to the bottom, separated by colon (e.g. top,person)
`ldap_group_dn`	The distinguished name of the group entries
`ldap_group_object`	The class hierarchy for group entries from the top to the bottom, separated by colon
`ldap_password_field`	The attribute names containing the password (separated by colon)
`ldap_user_field`	(optional - default=cn) The prefix for the unique userid

For more information about these parameters, see section ***LDAP Configuration***

Once a domain is defined in Tamino, any user accessing Tamino that has a username starting with the domain ID, followed by a backslash, will be authenticated against the authentication service specified by the domain. For example, if we have the following defined in *ino:security*:

```
<ino:domain xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
ino:domainid="mydomain" ino:domaintype="os" />
<ino:user xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
ino:userid="mydomain\user">OS-User</ino:user>
<ino:user xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
ino:userid="user" ino:unencryptedpassword="xx">Tamino-User</ino:user>
```

We can now access Tamino by specifying the username `user` without a domain, in which case the user will be authenticated against the password `xx`, which is stored directly in Tamino, or by specifying the username `user` and the domain `mydomain`, in which case the user will be authenticated against the local operating system.

Since the Tamino architecture requires the authentication information to be passed to the server in the HTTP header field for authentication, and since this field is restricted to a userid and password, the domain is represented as part of the userid separated by a backslash (“\”). Any userid which does not contain any domain information is automatically assigned to the default domain, which is the local Tamino authentication service, relying only on user information stored in *ino:user*.

LDAP Configuration

To configure an external LDAP service, some special parameters need to be set, depending on the type of LDAP being used. This section gives a short description for some common LDAP configurations.

iPlanet Directory Server

The iPlanet Directory Server is preconfigured to store users in a special way. When using this preconfigured mechanism, the following parameters should be used (besides the `host` and `port` parameters):

```
<ino:param ino:name="ldap_person_dn"
  ino:content="ou=People,dc=my,dc=server,dc=com" />
<ino:param ino:name="ldap_person_object"
  ino:content="top,person,organizationalPerson,inetorgperson" />
```

where the value `"ou=People,dc=my,dc=server,dc=com"` needs to be replaced by your specific DN.

If you have added users with the uid field filled, the users will not be uniquely identified using the `cn` prefix, but rather the `uid` prefix, which will require that the following parameter is also added:

```
<ino:param ino:name="ldap_user_field" ino:content="uid" />
```

Active Directory Service

When using an MS Active Directory Service, the following parameters need to be set (besides the `host` and `port` parameters):

```
<ino:param ino:name="ldap_person_dn"
  ino:content="CN=Users,DC=my,DC=server,DC=com" />
<ino:param ino:name="ldap_person_object"
  ino:content="top,person,organizationalPerson,user" />
```

where the value `"CN=Users,DC=my,DC=server,DC=com"` needs to be replaced by your specific DN.

OpenLDAP

An OpenLDAP server can be configured in various ways - the administrator of the LDAP will know which fields contain the user information, and should be able to provide the correct values for the fields `ldap_person_dn`, `ldap_person_object`, `ldap_password_field`, and possibly also the `ldap_user_field`.

Tamino Authorization

Tamino provides an authorization check to grant or deny access to XML nodes (elements and attributes) and functions.

Access control is currently available on the structural level, that is, each node can be protected individually, or based on its position in a subtree, rather than based on the content or value of a node.

This section describes how authorization checking is implemented in Tamino under the following headings:

- Authorization Concept
- Protecting ino:security
- Security Objects: XML Schemas
- Special Cases

Authorization Concept

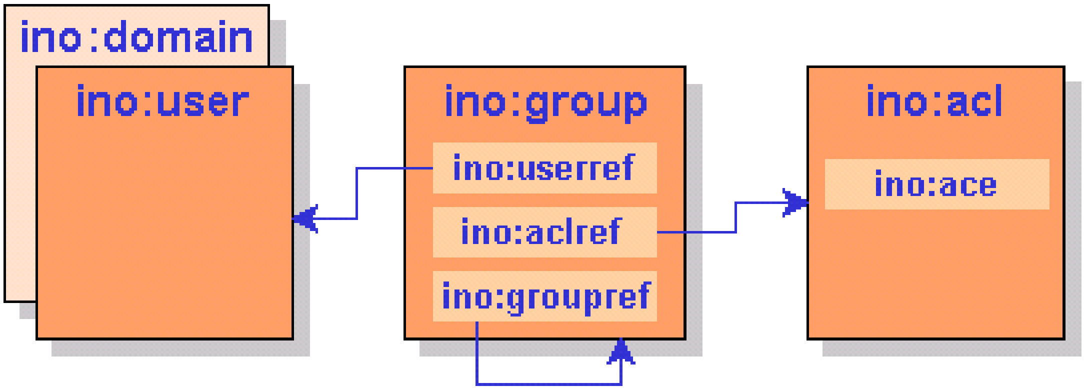
Security Objects

Tamino is delivered with a Collection *ino:security* that contains Doctypes *ino:user*, *ino:group* and *ino:acl* (acl=Access Control List).

Note: The Collection *ino:timeframe* is intended for future use and does not have special functions in the current version.

You can implement access control by defining instances of Tamino Doctypes *ino:user*, *ino:group* and *ino:acl* to the Tamino Collection *ino:security* as illustrated in the following figure:

Doctypes in Collection ino:security



An Access Control List (ACL) contains Access Control Elements (ACEs) which identify nodes and function names. The access level is specified in an attribute of the ACE. A group refers to users and ACLs, thus authorizing those users for the ACEs contained in the ACLs.

The authorization concept for nodes is based on the following principles:

- Each node to be protected must be XML content stored in Tamino. Nodes that are implicitly created due to mappings to external sources cannot be protected.
- Each node and its descendants can be protected individually, but if a given node cannot be accessed as a result of its access restrictions, then none of the descendants of that node can be accessed either, even if access permissions are defined for these descendants.
- If a node is not defined in an ACE, there is no access control, that is, it is generally accessible, unless there are restrictions defined for one of the node's ancestors.

The authorization concept for functions is based on the following principle:

- An absolute function name that is not preceded by any path names must be specified. The function name must finish with the characters ().
- The attribute ino:access for functions may only contain the values `no` (function may not be executed) or `full` (function may be executed).

The following general principles also apply:

- All undefined users are automatically members of the default group that has the same name as the database.
- Each instance of user, group, and ACL must be unique within a Tamino database.
- The authorization feature does not guarantee referential integrity. That is, if a node is deleted, the authorization information is retained, and restrictions can be defined for nodes before they exist.

- During document composition, protected nodes are not returned to an unauthorized user.
- During document processing, if the construct to be updated in Tamino contains protected nodes, the update is rejected for the whole document.
- During delete operations, if the construct to be deleted in Tamino contains protected nodes, the delete operation is rejected for the whole document, and also for multiple documents if more than one document is to be deleted.
- The order of precedence in which access restrictions are processed is such that less restrictive permissions override more restrictive ones if more than one restriction is explicitly defined for one node.

Authorization Levels

The authorization level is specified in an attribute of the ACE. Authorization levels are:

Level	Meaning
no	Access to this node or function is denied.
read	Read access is granted.
change	Update access is granted.
full	Define and undefine access is granted. Function may be executed.

Note: The access attributes are case-sensitive and must all be written in lower-case letters.

The following authorization levels are required on the Node for X-Machine commands:

Command	Minimum authorization level
_xql	read, change or full for node, and full for function
_process	change or full
_delete	change or full
_define	full
_undefine	full
_admin	full for function
_xquery (reading)	read, change or full
_xquery (updating)	change or full

Protecting ino:security

All authorization information in Tamino is stored in the Collection *ino:security*. When Tamino is initially installed, no access restrictions are set for this Collection, which means that any user authenticated by the web server to access Tamino can modify the data in ino:security and hence grant and revoke access rights to any part of the database. When implementing security on Tamino, you should first define a user that has access to *ino:security*, and define *ino:security* as being inaccessible to all other users.

The following XML document is an example of how to set up security for the Tamino database `mydb`, with only the user `secadmin` having access to *ino:security*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<ino:user ino:userid="secadmin"
   xmlns:ino="http://namespaces.softwareag.com/tamino/response2">Security ↩
Administrator</ino:user>

<ino:group ino:groupname="secgroup" ↩
xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:userref>secadmin</ino:userref>
  <ino:aclref>secacl</ino:aclref>
</ino:group>

<ino:acl ino:aclname="secacl" ↩
xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:ace ino:access="full">ino:security</ino:ace>
</ino:acl>

<ino:group ino:groupname="mydb" ↩
xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:aclref>defaultacl</ino:aclref>
</ino:group>

<ino:acl ino:aclname="defaultacl" ↩
xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:ace ino:access="no">ino:security</ino:ace>
</ino:acl>
```

Note: Please be aware that the user who is to be used to administer the security configuration, needs to be a valid user in the Tamino Server with the required privileges – otherwise it will not be possible to modify the user definitions.

Security Objects: XML Schemas

All security objects are stored in the Collection *ino:security*. Security objects are instances of Doctypes ***ino:user***, ***ino:group*** , ***ino:acl*** or ***ino:domain***. Please refer to the collection ino:collection, for the complete schemas.

Doctype ino:user

The attribute *ino:userid* is checked for uniqueness during insertion of an instance of Doctype *ino:user*. The contents of the *ino:user* element is freely definable and has no relevance to the operation of Tamino security.

Example ino:user Instances

The following examples illustrate an instance of a general user:

```
<ino:user ino:userid="saguser1" ino:password="xxx" ↩
xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
User name, general user</ino:user>
```

Doctype ino:group

The Doctype *ino:group* has only one attribute: *ino:groupname*. The group name is checked for uniqueness during insertion of an instance of Doctype *ino:group*.

The child element *ino:userref* establishes the relation between an instance of *ino:group* and an instance of *ino:user* by using the value of the user ID as content.

On the other hand, there is no referential integrity between *ino:user* and *ino:group* based on the contents of the *ino:userref* element.

The child element *ino:aclref* establishes the relation between an instance of *ino:group* and an instance of *ino:acl* by using the name of the ACL as content. All users referenced in the group have access to all ACLs referenced by the same group. Again, there is no referential integrity between *ino:group* and *ino:acl* based on the contents of the *ino:aclref* element.

Using the *ino:groupref* element, it is possible to reference other groups. The references are resolved similar to an `include` statement.

The following example demonstrates how the *ino:groupref* can be used to separate the groups of users and groups of privileges (roles). The *adminusers* group is used to group some users together and the group *adminprivileges* groups some ACLs. Via the *ino:groupref* from adminusers to *adminprivileges*, the users in group *adminusers* are assigned the ACLs from *adminprivileges*.

```
<ino:group ino:groupname="adminusers"
  xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:userref>saguser1</ino:userref>
  <ino:userref>saguser2</ino:userref>
  <ino:groupref>adminprivilege</ino:groupref>
</ino:group>

<ino:group ino:groupname="adminprivilege"
  xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:aclref>acldbadmin</ino:aclref>
  <ino:aclref>aclsecadmin</ino:aclref>
</ino:group>
```

Special Groups

If there is a group defined to the system, which has a group name identical to the name of the Tamino database in question, all defined and undefined users are regarded as members of this group.

Besides the default group, there is also a predefined administrator group with the name *ino:admin*. Any user assigned to this group has full access to all nodes and to all functions.

Example ino:group instances

The following example instances illustrate a group for general users:

```
<ino:group ino:groupname="general" ↩
xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:userref>saguser1</ino:userref>
  <ino:aclref>aclgeneral</ino:aclref>
</ino:group>
```

The following example instances illustrate a group for administrators:

```
<ino:group ino:groupname="admin" ↩
xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:userref>sagdba</ino:userref>
  <ino:aclref>acladmin</ino:aclref>
</ino:group>
```

The following example instances illustrate a default group for a database that is called mydb:

```
<ino:group ino:groupname="mydb" ↩
xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:aclref>aclall</ino:aclref>
</ino:group>
```

The group definition *mydb* will automatically incorporate all users of the Tamino database, because it carries the name of the Tamino server itself (in this example *mydb*), thus granting the access rights contained in the ACLs referenced in *ino:aclref* to all users.

Note: This group has to be created to act as default to prevent any user getting full database access.

Doctype ino:acl

The *ino:acl* element has only one attribute: *ino:aclname*. The ACL name is checked for uniqueness during the insertion of an instance of Doctype *ino:acl*. The child elements *ino:ace* contain the access authorization definitions for each node and function in the Tamino repository as values of the attribute *ino:access*.

The content of an *ino:ace* element is the absolute path to the node or the absolute function name for which access rights are to be defined. The path must be explicitly defined, it must not contain any generic definitions such as // or *. Also, the names between the slashes may only consist of characters that are valid for XML element and attribute names.

There are currently four possible values for the attribute *ino:access*: no, read, change, full. Any other value supplied will be rejected by the validation. Note that these values are case-sensitive.

Example Instances of ino:acl

This example grants read access to instances of `doctype1` in collection `coll1`.

```
<ino:acl ino:aclname="aclall" ↩
xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:ace ino:access="read">coll1/doctype1</ino:ace>
</ino:acl>
```

This example grants full access to instances of "doctype1" in Collection "coll1"

```
<ino:acl ino:aclname="acluser" ↩
xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:ace ino:access="full">coll1/doctype1</ino:ace>
</ino:acl>
```

This example grants full access to the Collection *ino:security* (for administrators):

```
<ino:acl ino:aclname="acladmin" ↩
xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:ace ino:access="full">ino:security</ino:ace>
</ino:acl>
```

This example grants execution access to the function *ino:Info*:

```
<ino:acl ino:aclname="acladmin" ↩
xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
<ino:ace ino:access="full">
ino:Info()</ino:ace>
</ino:acl>
```

Doctype ino:domain

Example Instances of ino:domain

This example defines a domain *mydomain*, for which all users will be authenticated against the local operating system:

```
<ino:domain xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
ino:domainid="mydomain" ino:domaintype="os"/>
```

Special Cases

This section describes the behavior of Tamino Security in some special situations.

_process

In order to insert data using the `_process` command, the permissions `full` or `change` are required for all nodes into which data will be inserted. If data is to be stored in a Collection in which a restriction of `no` or `read` has been defined on a required node for a user, the store process will fail even if no data is to be stored in the node that is non-accessible. This is because of the validation, which will not accept the document as it doesn't conform to the schema definition.

_delete

In order to delete a document, a user must have at least change permission on all of the nodes that are to be deleted. If more than one document is to be deleted in a single request and at least one document cannot be deleted because of access restrictions, then the entire delete request will fail.

Restricting required nodes

When setting access restrictions on nodes, care should be taken when restrictions are set on nodes that are not optional. When data is read from such documents, the inaccessible nodes are not returned in the result set. However, it is not possible to load new data into these Doctypes because the validation will not accept data that does not conform to the schema definition, and data conforming to the schema definition will be rejected by security because inaccessible nodes are involved.

Error messages

Tamino does not issue error messages that provide detailed insight into the exact cause of access violations. This is done with the intention of providing potential hackers with as little information as possible. When a user attempts to read data to which she/he has no access, security normally returns the same information that it would as if the protected data did not exist. When attempts are made to delete or modify protected data, Tamino issues a general access violation message in most cases, but does not provide any information about the exact cause of the violation.

Mass loading data

If you mass load data into the collection *ino:security*, it is strongly recommended that you use the option `concurrentWrite`. If this option is not used, the collection *ino:security* will be exclusively locked by the mass loader, and no other incoming requests will be authenticated or authorized until the mass load process is finished.

WebDAV

Note that with the introduction of WebDAV as a built-in component of the Tamino XML Server, a new type of security is also available in the WebDAV perspective. While the Tamino Security mainly addresses the security of parts of XML documents based on their structure, the WebDAV ACL Specification (RFC 3744) addresses protecting operations on whole documents only. For details, refer to the WebDAV documentation.

Security Manager

You can use the Security Manager to create, display, delete and update users, user groups and access control lists for the selected database. The database selected must be running. The following Security Manager functions are available:

- Maintain Users
- Maintain User Groups
- Maintain Access Control List

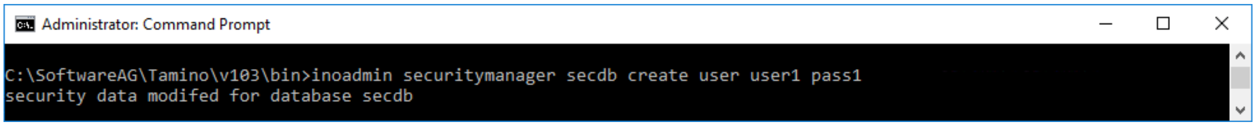
Notes:

1. For the names of users, groups, ACLs and ACEs, the full UTF-8 character set is supported.
2. If the "Authentication" property is set to "tamino", i.e. access to the database requires authorization, user and password must be specified using the environment variables INOADMIN_SECMAN_USER and INOADMIN_SECMAN_PWD.

Maintain Users

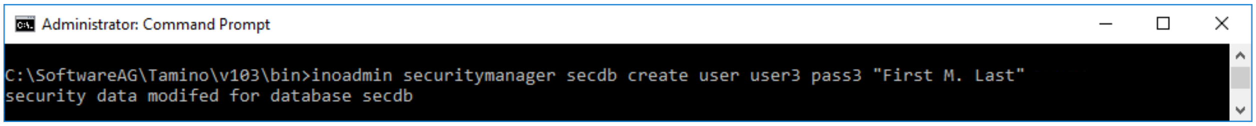
Creating a user means to create (define) an instance of the Doctype *ino:user*. Once a user is created it may become a member of one or more groups.

A user is created via the **inoadmin securitymanager** command as follows:



```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb create user user1 pass1
security data modifed for database secdb
```

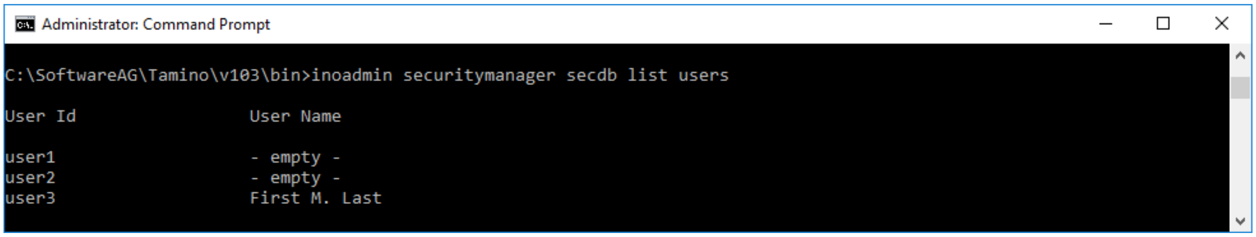
An additional, optional parameter may be used to assign a description to the user object.



```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb create user user3 pass3 "First M. Last"
security data modifed for database secdb
```

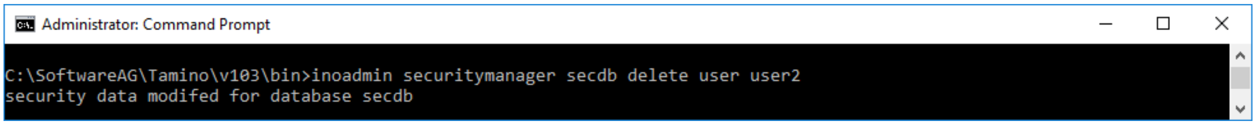
Apart from being a member of one or more groups, users can also become members of a domain (only one).

The **inoadmin securitymanager** command **list users** lists all users.



```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb list users

User Id                 User Name

user1                   - empty -
user2                   - empty -
user3                   First M. Last
```

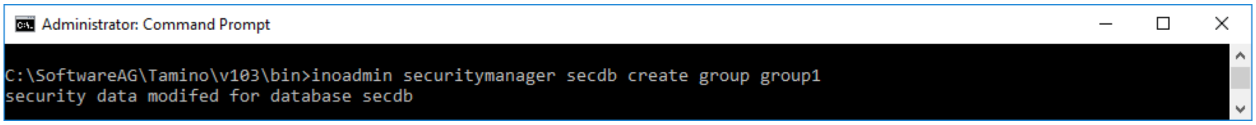
The **delete user** command deletes a selected user (instance of Doctype *ino:user*) from the database. If you only want to remove a user from a specific group, you should use the **remove user** command.



```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb delete user user2
security data modifed for database secdb
```

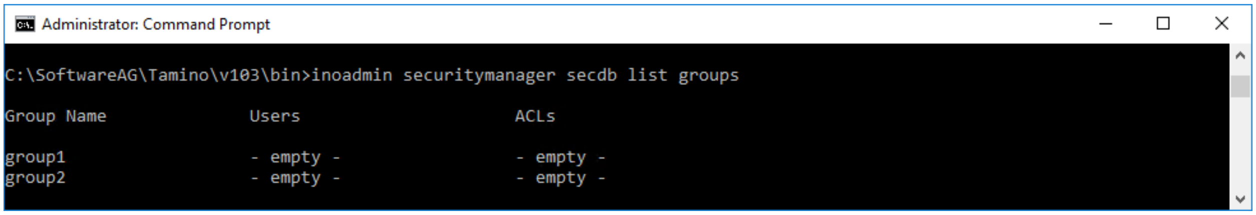
Maintain User Groups

Creating a Group means to create an instance of Doctype *ino:group*. Once created, existing users and access control lists can be added to the new group by using the **add user** and **add acl** command respectively.



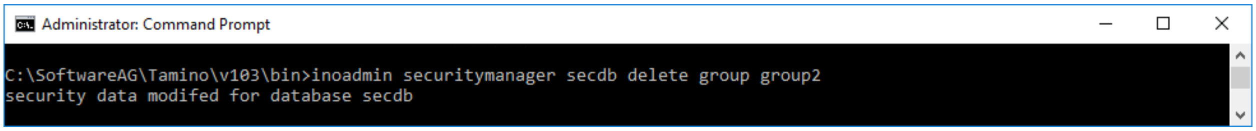
```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb create group group1
security data modifed for database secdb
```

All existing user groups can be listed using the **list groups** command.



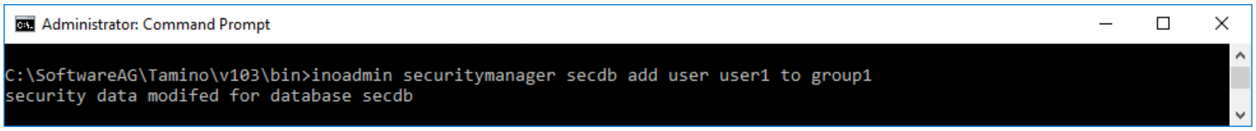
```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb list groups

Group Name              Users                    ACLs

group1                  - empty -                - empty -
group2                  - empty -                - empty -
```

The **delete group** command deletes a user group (instance of Doctype *ino:group*). However, the users and ACLs that are members of that group are not deleted.



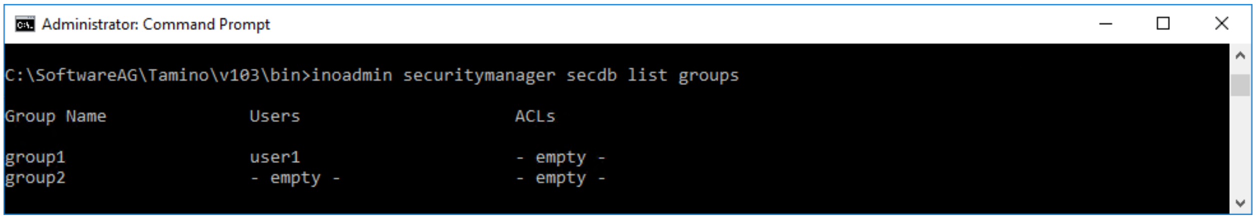
```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb delete group group2
security data modifed for database secdb
```

Users are added to groups via **add user**.



```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb add user user1 to group1
security data modifed for database secdb
```

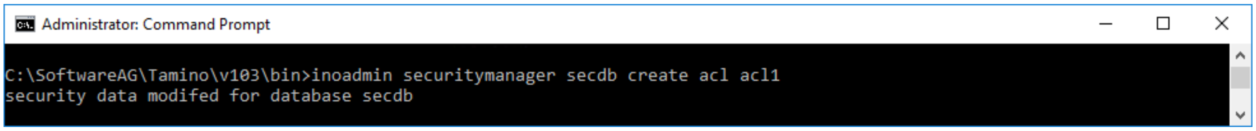
The **list groups** command now also lists the group's user.



```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb list groups

Group Name              Users                    ACLs

group1                  user1                    - empty -
group2                  - empty -                - empty -
```

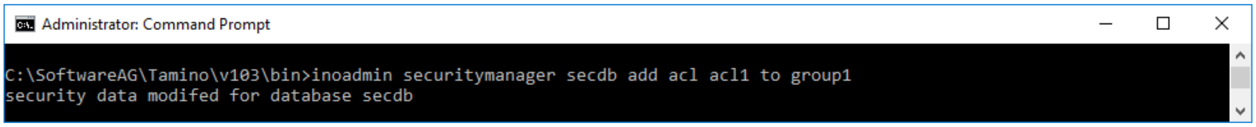
Maintain Access Control List

The **create acl** command is used to create an instance of the Doctype *ino:acl* (access control list).



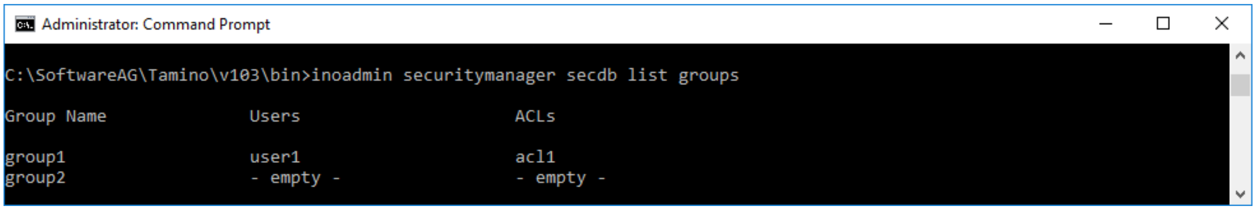
```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb create acl acl1
security data modifed for database secdb
```

If groups already exist, the new ACL can be added to a group. An ACL can be a member of one or more groups.



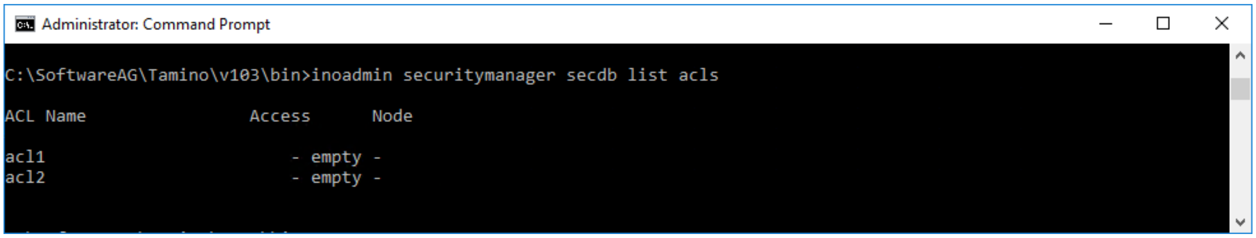
```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb add acl acl1 to group1
security data modifed for database secdb
```

The **list groups** command now also lists the group's ACLs.



```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb list groups

Group Name              Users                     ACLs

group1                  user1                     acl1
group2                  - empty -                 - empty -
```

The **list acls** command is used to show all ACLs.



```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb list acls

ACL Name                Access      Node

acl1                        - empty -
acl2                        - empty -
```

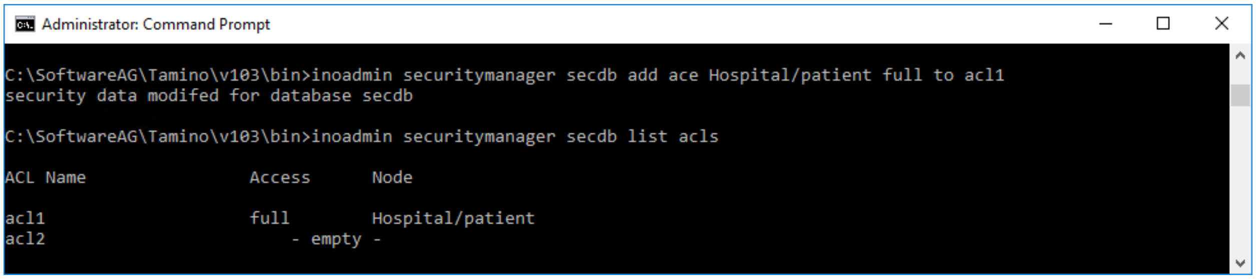
An ACL is meant to hold a collection of access control elements (ACEs) that each describe access rights upon specific parts of XML documents in the Tamino repository. The definitions are stored in the child elements *ino:ace* of the Doctype *ino:acl*. The access control element is added to the access control list that is to be given access rights to the node specified.

To add an access control element use the **add ace** command and name the node to which access rights are to be defined (collection/doctype etc.) and the access level. There are four access levels available:

Level	Meaning
no	Access to this node is denied.
read	Read access is granted.

Level	Meaning
change	Update access is granted.
full	Define and undefine access is granted.

Having added ACEs to ACLs the ACEs are listed in the **list acls** command.



```
C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb add ace Hospital/patient full to acl1
security data modifed for database secdb

C:\SoftwareAG\Tamino\v103\bin>inoadmin securitymanager secdb list acls

ACL Name                Access      Node

acl1                    full        Hospital/patient
acl2                        - empty -
```

The **delete acl** command deletes a selected access control list and removes references to it from the groups in which it was contained. If you only want to remove an ACL from specified groups, you should use the **remove acl** command.

Notes:

1. When using the Security Manager, the user must have the privileges to make changes to *ino:security*. This means that an administrator must be defined in Tamino before the Authentication mode is switched to `tamino`.
2. The feature of using the local operating system for authentication cannot currently be configured using the Security Manager. This means the *ino:domain* definitions have to be stored directly into Tamino using other interfaces (e.g. with the Tamino Interactive Interface).

Instance-based Security

Tamino provides not only type-based access control but also instance-based acceeee control . With instance-based access control, it is also possible to define access control for each document, thereby enabling Tamino as a directly accessible document repository in a multi-user environment.

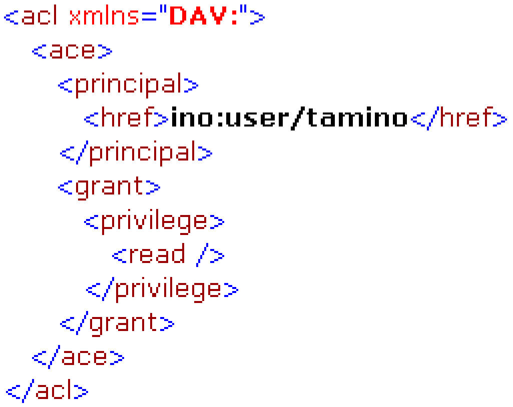
The following description gives an overview of the privileges defined for the instance-based access control and shows how the ACLs are defined and read.

Access Control Lists

Since WebDAV ACL defines an XML structure for access control lists and since Tamino also supports WebDAV ACL for the WebDAV resources stored in Tamino, this ACL structure was also chosen for regular Tamino documents.

An access control list consists of one or more ACEs (access control elements) which each define a set of privileges to be granted or denied to a certain principal. Per ACE there can be multiple privileges defined, but they must be either all granted or all denied.

The following gives an example of an ACL granting the user *tamino* read access to a document:

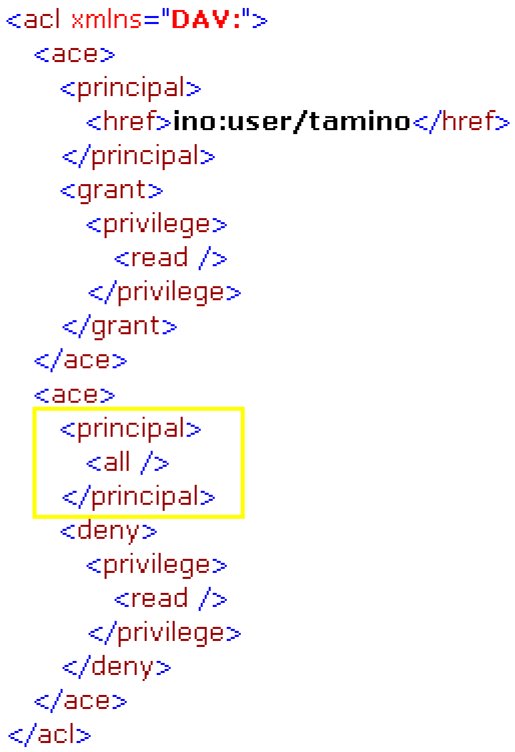


```
<acl xmlns="DAV:">
  <ace>
    <principal>
      <href>ino:user/tamino</href>
    </principal>
    <grant>
      <privilege>
        <read />
      </privilege>
    </grant>
  </ace>
</acl>
```

Principals

Principals can be basically any users and groups known to the Tamino database. When addressing a certain user or group the href element needs to be specified and the content of that element denotes the principal by specifying the name preceded by the string "ino:user/" for users and "ino:group/" for groups.

Besides the explicit definition of users and groups there is also the possibility to use the special principal all to apply to all users. The following ACL gives an example which would grant the user tamino read access to a document and deny the read access to all other users:



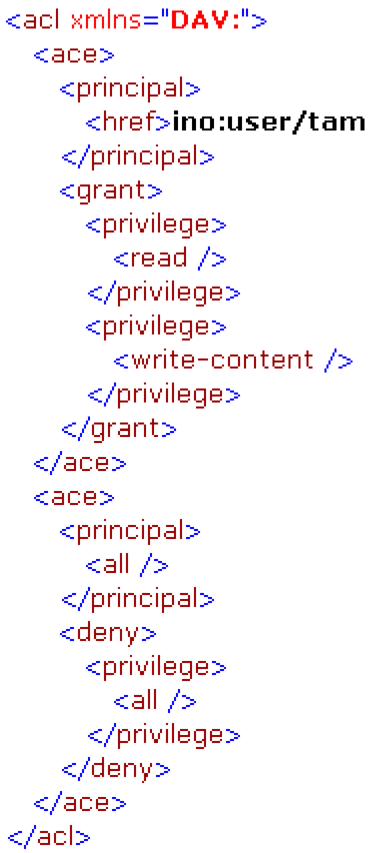
```
<acl xmlns="DAV:">
  <ace>
    <principal>
      <href>ino:user/tamino</href>
    </principal>
    <grant>
      <privilege>
        <read />
      </privilege>
    </grant>
  </ace>
  <ace>
    <principal>
      <all />
    </principal>
    <deny>
      <privilege>
        <read />
      </privilege>
    </deny>
  </ace>
</acl>
```

Privileges

The WebDAV ACL standard defines a set of privileges to control the access to certain operations on the content and properties of documents. The most commonly used are:

- **write-content**
  Controls whether a user is allowed to modify the content of a document.
- **read**
  Controls whether a user is allowed to read a document.

According to the WebDAV ACL standard the privileges can also be aggregated. One predefined aggregation is the privilege all which contains all other privileges. The following ACL grants read and write-content access to the user tamino and denies everything for all other users:



```
<acl xmlns="DAV:">
  <ace>
    <principal>
      <href>ino:user/tamino</href>
    </principal>
    <grant>
      <privilege>
        <read />
      </privilege>
      <privilege>
        <write-content />
      </privilege>
    </grant>
  </ace>
  <ace>
    <principal>
      <all />
    </principal>
    <deny>
      <privilege>
        <all />
      </privilege>
    </deny>
  </ace>
</acl>
```

Besides the predefined privileges, an ACL may also contain user-defined privileges which don't have any semantics for Tamino but might be used by external applications to control access to certain operations on the documents.

Evaluation of ACLs

The ordering of the ACEs in an ACL is important, since the ACLs are evaluated from top to bottom. When the access level to a document is determined, the Tamino server will search for an ACE entry which matches the current request principal (either the user itself or a group to which the user belongs) and which also matches the required privilege. So if a user wishes to read a document, the ACL is scanned for an ACE which matches the principal and contains the privilege read. If that privilege is granted, the operation will be executed. If the privilege is denied, the execution of the operation will be rejected. If the whole ACL does not contain any ACE matching the principal and the required privilege, a grant of the privilege will be applied and the operation will be executed – this means that a good ACL will typically end in a deny all for everyone.

Defining an Access Control List

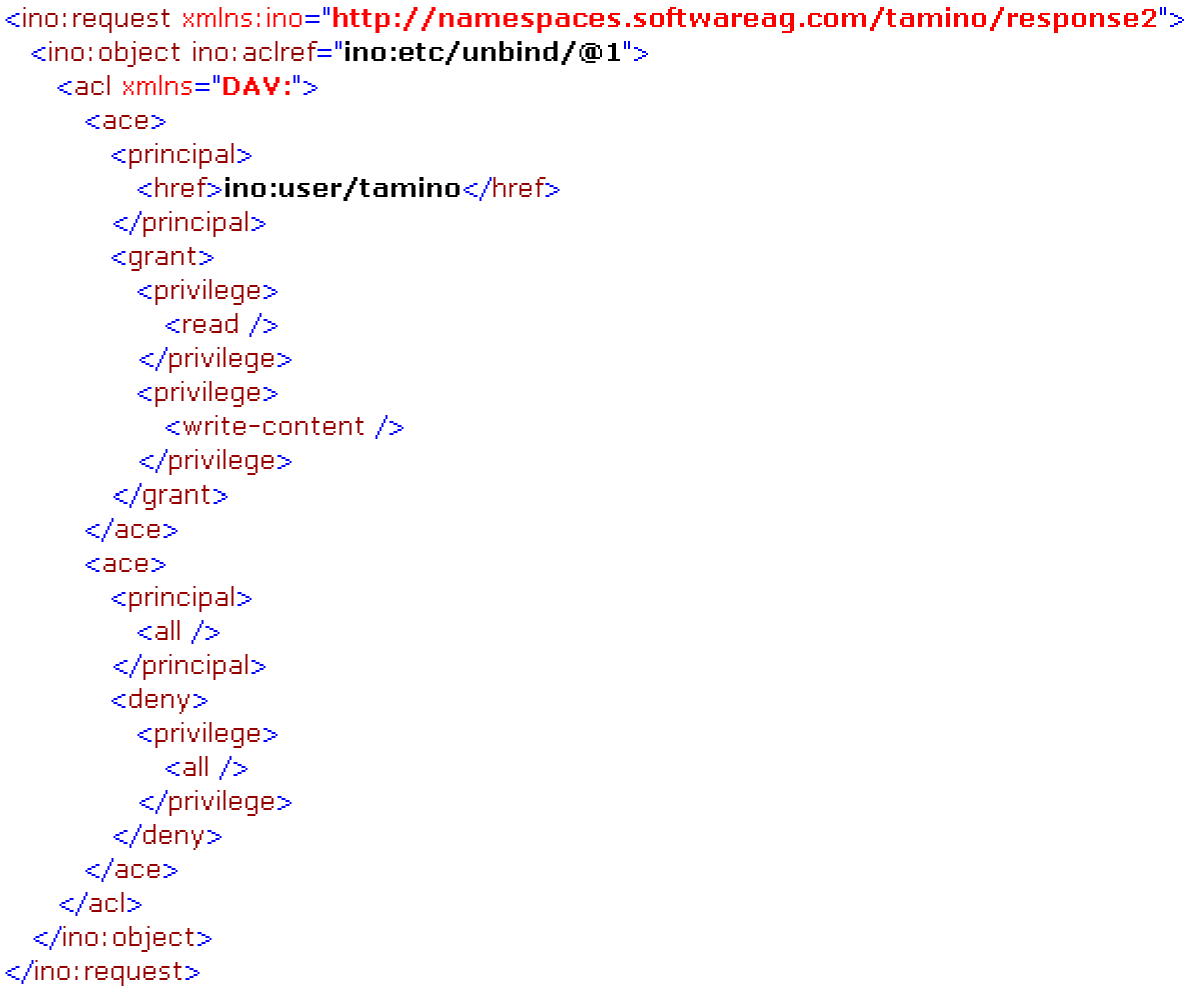
An ACL is provided as an XML document and can be applied to a Tamino document in two ways: either via a PROCESS where the ACL is wrapped with a special element defining the document to apply this ACL to or via XQuery when creating a new document.

Defining an ACL via the PROCESS command

To apply an ACL to a document, this document has to be uniquely identified. An ACL can be wrapped by an `ino:request` and `ino:object` element and a reference to the document to which this ACL should apply can be set via the special attribute `ino:aclref`. The reference is a path of the following structure:

```
collection name + '/' + doctype name + '/@' + ino:id
```

So to define an ACL for the document with `ino:id` 1 in the collection ino:etc and in the doctype with name unbind, the following document would be stored via a PROCESS command:

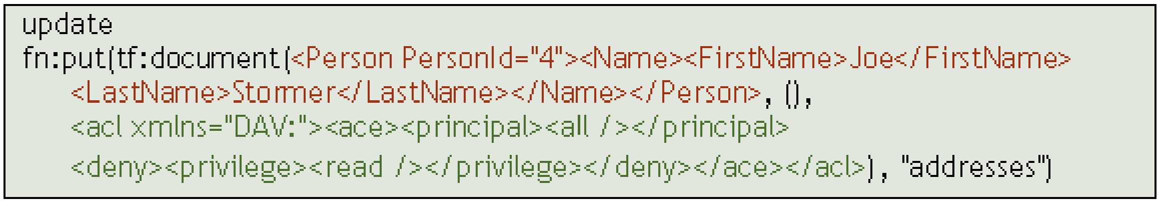


```
<ino:request xmlns:ino="http://namespaces.softwareag.com/tamino/response2">
  <ino:object ino:aclref="ino:etc/unbind/@1">
    <acl xmlns="DAV:">
      <ace>
        <principal>
          <href>ino:user/tamino</href>
        </principal>
        <grant>
          <privilege>
            <read />
          </privilege>
          <privilege>
            <write-content />
          </privilege>
        </grant>
      </ace>
      <ace>
        <principal>
          <all />
        </principal>
        <deny>
          <privilege>
            <all />
          </privilege>
        </deny>
      </ace>
    </acl>
  </ino:object>
</ino:request>
```

The document needs to be sent to the collection *ino:security*. The ACL will not be physically stored in the collection *ino:security*, but will rather be attached directly to the document to which the ACL applies.

Defining an ACL via XQuery

Another mechanism for setting an ACL for a document is to directly attach the ACL to a newly created document via XQuery. The XQuery function tf:document allows you to create a new document and append an ACL to the document. When this document is then stored to Tamino via the fn:put function, the ACL will be directly applied to the document. The following example shows how a document with root element `Person` is created and saved into the collection addresses while being assigned an ACL that denies the read privilege to everyone:



```
update
fn:put(tf:document(<Person PersonId="4"><Name><FirstName>Joe</FirstName>
   <LastName>Stormer</LastName></Name></Person>, (),
   <acl xmlns="DAV:"><ace><principal><all /></principal>
   <deny><privilege><read /></privilege></deny></ace></acl>), "addresses")
```

Instance-based vs. structure-based Access Control

When both instance-based and structure-based access control are defined for documents, the privileges for both types of access control need to be granted in order for the operation to be allowed. The combination of the two types of control can be quite useful – for example it allows you to set the access level for a certain group of users to read permissions for a whole collection via structure-level access control and restrict the access even more to make some of the objects not readable at all for this group of users.

The special group *ino:admin*, which allows full access to all data, also applies to the instance-based access control. So if a certain privilege is denied for everyone in an instance-based ACL, the members of the *ino:admin* group will still get full access to the document.

5 Administration Considerations

This section provides general information about the following topics:

- Space management
- Problem reporting

Space Management

Database Locations

In order to use the physical storage on a node, the Tamino Manager uses the concept of database locations. Database locations are where the Tamino database spaces are stored. The Tamino Manager maps these database locations (which can also be thought of as logical names) to physical location paths.

Let us suppose that a database administrator has a directory called C:\Myfiles\Tamino\mydb-backup, in which she/he plans to store backup copies of his database mydb. Using the Database Location Manager, she/he can assign the database location name of mydbBackup to this directory. Once this has been done, she/he can administrate the files via the location name without having to remember where they are physically located.

Database Spaces

Database spaces are the physical components that comprise the database. They are stored in and administrated via database locations.

Types of database spaces

The following types of database spaces exist for each database:

Space Type	File Extension	Purpose
Index Space	1I0	Contains the database's meta data, e.g. indexes
Data Space	1D0	Contains the database's data
Log Space	1L0	Contains the information required to recover updates after restoring a database
Backup Space	1B0	Contains database backups
Tamino CD Database Info File	1R0	Contains the information required to register a CD database with the Tamino Manager
Journal Space	1J0	Contains the information required for transaction rollback
Journal Overflow Space	1O0	Journal overflow spaces are used to save pending global transactions and the logging records of long running transactions. Journal overflow is automatically created or

Space Type	File Extension	Purpose
		extended if it is necessary, and it will also be automatically removed or decreased during server shutdown.
Temporary Sort Space	1S0, 1T0, 1F0, 1W0	Contains temporary data
Utility Recovery Space	1C0, 1C1	Contains log information from calls to the Data Loader
Tamino dump file	0M0, 1M0	Contains information for error analysis by Software AG support

Note: Do not directly manipulate these files, but use the **inoadmin** command line tool for all operations. These files are deleted when a database is deleted, with the exception of the dump files. Dump files can be deleted manually if they are no longer needed for error analysis by Software AG support. Backup files can be kept with a corresponding option in the **Delete Database** dialog box.

Problem Reporting

How to prepare Problem Information

If a problem occurs at a customer's site, Software AG's support team will attempt to provide a solution as fast as possible (in some cases, software is also distributed via partners, who are responsible for support themselves). To minimize the time required to analyze a problem and deliver a solution, the support team requires detailed information about the problem.

The problem information should contain:

- Problem description:
  - Tamino component or function used
  - Error messages (message IDs) and texts
  - A short description of the problem
- Environment description:
  - Tamino version
  - Operating system version
  - Hardware environment (CPU type, disk device types, etc.)
- Can the problem be reproduced?
  - What are the steps to reproduce it
  - Which data is needed to reproduce it (schemas, XML instances, queries)

If the problem is not reproducible, the problem information should contain additionally:

- Where does the problem occur?
  - On all databases, or on a single database only
  - On databases within a specific environment only
  - Environments where the problem does or does not occur
- When does the problem occur?
  - Always or only sporadically
  - In parallel with other events
  - Periods of time when the problem does not occur
  - When the problem first occurred
  - Changes of the environment around this date
  - Last changes of the environment before this date

If Tamino creates crash dumps that are associated with a problem, they can also be sent to the support team. The files in question have the suffix 0M0 or 1M0 and can be found in the database's reserve location. The files should be compressed using a ZIP utility before sending them in order to save space.

Index