inxmail PROFESSIONAL 4

Inxmail Professional Manual

Developer Guide for PHP

Inxmail Professional API 1.13.1

Contact address

Phone: +49 761 296979-0
Email: info@inxmail.de

Find out more about Inxmail GmbH and the email marketing solution Inxmail Professional at www.inxmail.com

This document describes how to install use the Inxmail API. This is a technical paper. Knowledge of the chosen operating system and of programming in the Java, PHP or .NET[1] is required.

[1] Java is a registered trademark of Oracle Inc.
.NET is a registered trademark of Microsoft Inc.

Contents

1. Change History

Version 1.0.0 of the Inxmail API was introduced June 2005 with Inxmail Professional 3.2. Since then it has undergone some changes, most of them introducing new features to make more functionality of Inxmail available through the API.

1.1. Inxmail API 1.13.1

Changes in API 1.13.1, since Inxmail Professional 4.4.2

- **New: Access to split test and split test mailing objects**
  The new SplitTestManager and SplitTestMailingManager provide an interface which can be used to aggregate all split test mailings that refer to the same split test. For more information, see section Inx_Api_SplitTest_SplitTestManager and Inx_Api_SplitTestMailing_SplitTestMailingManager
- **New: Access to data source transformations**
  The new TransformationManager provides access to the data source transformations used in the Inxmail Professional content agent. It can be used to find, create and delete transformations as well as editing the XSLT code. For more information, see section Inx_Api_Transformation_-TransformationManager.
- **New: Bounce incorporates sending ID**
  The Bounce object now incorporates the sending ID, if available. This allows the correlation between a bounce and a sending which is especially interesting in case of trigger mailings, because these mailings may be sent multiple times.
- **Bugfix: Creation of recipients does not work - PHP**
  Inxmail Professional API version 1.11.10 introduced a bug that prevented the creation of recipients.
- **Bugfix: Creation of test recipients does not work - PHP**
  Inxmail Professional API version 1.11.10 introduced a bug that prevented the creation of test recipients.
- **Bugfix: new http option 'enableSNI' - PHP**
  New option 'http.enableSNI' solves connection problems in SSL and Proxy contexts

1.2. Inxmail API 1.12.1

Changes in API 1.12.1, since Inxmail Professional 4.4.1.223

- **New: Fluent interface for link data queries**
  To ease the creation of complex link data queries, this version of the Inxmail Professional API introduces a new fluent interface which can be used to create and execute such queries.
- **New: Direct connection between sending history and clicks**
  Now the sending object allows to determine the corresponding click data and vice versa.
- **New: Fluent interface for bounce queries and filter by bounce category**
  To ease the creation of complex bounce queries, this version of the Inxmail Professional API introduces a new fluent interface which can be used to create and execute such queries. It also provides the possibility to filter by bounce category.

- **New: Support for spam bounces and auto responder bounces**
  All bounce result sets will include bounces of category spam and/or auto responder if they match the queried filter.
- **New: Subscription log includes sending id**
  Entries of the subscription log include the related sending id if applicable.
- **Bugfix: Unique action names are enforced - All languages**
  As of Inxmail Professional 4.4.1, creating an action with the same name as an existing action will cause an UpdateException to be thrown on commit. Updating an existing action to a new name that is already in use also triggers an UpdateException.

1.3. Inxmail API 1.11.10

Changes in API 1.11.10, since Inxmail Professional 4.4.0.900

- **New: Standardised access to most mailing types**
  The new GeneralMailingManager provides read-only access to most of the mailing types supported by Inxmail Professional through a single interface. For more information, see section Inx_Api_GeneralMailing_GeneralMailingManager.
- **New: Restricting link retrieval to permanent links**
  The LinkData class now offers methods to restrict the result set to contain permanent links only. For more information refer to section *LinkData* in chapter *Inx_Api_DataAccess_DataAccess*.

Bugfix: LinkDataRowSet does not contain temporary links - All languages
The previous version of the API contained a bugfix which removed temporary links from the LinkDataRowSet. This caused problems in some applications based on the Inxmail Professional API. Therefore, the default behavior was switched back to including temporary links, too.

- **Bugfix: NullPointerException when parsing mailings with non-existing sending ID - All languages**
  In previous versions of the API, an attempt to parse a mailing with a non-existing sending ID using one of the renderers caused a server side NullPointerException. This is no longer the case, instead the invalid sending ID is ignored.

1.4. Inxmail API 1.11.5

Changes in API 1.11.5, since Inxmail Professional 4.4.0.820

Note: Performance improvements in SendingHistoryManager
The performance of the SendingHistoryManager was increased dramatically. For more information on the performance characteristics of the SendingHistoryManager, see section *Performance considerations* in chapter *Inx_Api_Sending_SendingHistoryManager*.

Bugfix: Huge ClickDataQuery can cause server crash
In beta API version 1.11.4 it was possible to trigger a server crash by fetching a huge amount of clicks using the fluent click interface. To prevent this issue, the maximum number of clicks which can be retrieved in a single query has been limited. For more information on this topic, see section *Performance considerations* in chapter *Fluent interface for clicks*.

- **Bugfix: RecipientContext.findByIds no longer filters non existent recipients**
  The findByIds method of the RecipientContext no longer filters non existent recipients, instead triggering a DataException when trying to access a recipient record which is not existing.

- **Bugfix: RecipientRowSet.getId throws NullPointerException if recipient does not exist**
  In API version 1.11.5, method getId of class RecipientRowSet no longer throws a NullPointerException if the current recipient does not exist, but instead throws a DataException as is documented in the code documentation of this method.

1.5. Inxmail API 1.11.4 (Beta version)

Changes in API 1.11.4 (Beta version), since Inxmail Professional 4.4.0.705

- **New: Access to sending history**
  The new SendingHistoryManager provides access to information regarding the sending of mailings, including open/click count, recipient reactions and other sending statistics.
- **New: Generics, Iterable and Closable - Java**
  This version of the Inxmail Professional API was completely revised to use Generics. In addition, BOResultSets and RecipientMetaData now implement Iterable for easy iteration using the for-each loop. Furthermore, Java 7 users will probably like the possibility to use the try-with-resources statement on any API class which provides a close method. All these refactorings provide a much cleaner and more concise way of using the Inxmail Professional API.
- **New: BOResultSets implement IEnumerable - .NET**
  In the new version of the Inxmail Professional API for .NET, BOResultSets implement the IEnumerable interface for easy iteration using the for-each loop.
- **New: BOResultSets implement Iterator - PHP**
  As in the Java and .NET versions of the Inxmail Professional API, the PHP version implements Iterator for easy iteration using the for-each loop.
- **New: Fluent interface for click data queries** To ease the creation of complex click data queries, this version of the Inxmail Professional API introduces a new fluent interface which can be used to create and execute such queries.
- **New: Configurable batch size for clicks**
  To provide a better means of performance tuning you can now specify the batch size for click data on a per-request basis. This allows you to control how many data records are transferred with each server call.
- **New: Improved access to unsubscription date**
  The UnsubscriptionRecipientRowSet now provides a more intuitive means of accessing the unsubscription date using the getUnsubscriptionDate() method.
- **New: Configurable timeout for server calls**
  In some cases a server call takes so much time that it is aborted. This may be due to a slow network connection, large amounts of data being transferred or - in a worst case scenario - a combination of both. If you experience this problem on a regular basis, you can now specify the read timeout on session creation.
- **New: Access to the connection URL**
  In some special cases you may want to know the connection URL a session was created with. This is now possible using the getConnectionUrl() method.
- **Note: New reports in Reports Reference**
  The Reports Reference in the appendix of the manual was updated with various new reports, including trigger mailing reports.
- **Note: Increased session timeout and click data batch size**
  With Inxmail Professional 4.4, the default session timeout was increased from six minutes to

nine minutes. The default batch size for click data was increased from 50 to 500, which should dramatically increase the performance of the click data retrieval.

Bugfix: Incorrect behaviour when deleting a single row - .NET
When deleting a single row in a row set or BOResultSet, the .NET version of the Inxmail Professional API always deleted the first row in this row set or BOResultSet. If you do delete single rows using the Inxmail Professional API for .NET, we strongly recommend updating to this version.

- **Bugfix: Personalization with test profiles is impossible - .NET**
  Prior to version 1.11.4 of the Inxmail Professional API it was impossible to personalize a mailing with a test profile instead of a recipient.
- **Bugfix: Retrieval of DesignCollections and Mailings - .NET**
  Version 1.9.0 of the Inxmail Professional API for .NET introduced a bug that prevented the retrieval of DesignCollections and Mailings.
- **Bugifx: Mailing cannot be locked using Hessian - .NET**
  Using the Hessian protocol, it was impossible to lock Mailings. This bug only occurred in the .NET version of the Inxmail Professional API.
- **Bugfix: Closing a session might trigger a fatal error - .NET**
  In some cases (bad timing), closing a session could trigger a fatal error, thus aborting program execution. This bug also only occurred in the .NET version of the Inxmail Professional API.
- **Bugfix: Possible double subscription/unsubscription - PHP**
  Under certain circumstances, using the processSubscription and processUnsubscription functions of the SubscriptionManager caused a double subscription/unsubscription. While this did not cause any problems in regard to the recipient's state, registered actions might have been triggered twice.
- **Bugfix: Retrieval of old web pages causes SOAP error - All languages**
  Prior to Inxmail Professional API version 1.11.4, web pages which lacked either a creation date or a sub type caused a SOAP error on retrieval, stating that these attributes may not be null. Wep pages created with Inxmail Professional 4.1 or lower had no creation date.
- **Bugfix: Exception on findByKey - All languages**
  Calling the findByKey(String) method of the RecipientContext caused a server-side NullPointerException if the given recipient was unknown.
- **Bugfix: TestRecipientRowSet always empty after deleteRow(s) - All languages**
  Due to a server bug, a TestRecipientRowSet was always empty after a call to deleteRow or deleteRows.
- **Bugfix: LinkDataRowSet contains temporary links - All languages**
  Starting with Inxmail Professional 4.4, LinkDataRowSets no longer contain temporary links, as this is in most cases not intended.
- **Bugfix: Forced unlocking impossible - All languages**
  Inxmail Professional 4.2 introduced a bug which prevented the unlocking of business objects which were locked by other sessions.
- **Bugfix: Invalid list ID allowed for approver - All languages**
  Prior to Inxmail Profession API version 1.11.4, zero was considered a valid list ID during the creation of approvers. Because the approver was therefore list specific but not bound to any existing list, the approver could neither be used nor recreated.
- **Bugfix: Invalid values allowed for SetValueCommand - All languages**
  Prior to Inxmail Professional 4.4, some invalid values could be used for the SetValueCommand which caused an invalid state.

- **Bugfix: Missing/incorrect security checks - All languages**
  Prior to the latest version of Inxmail Professional 4.3, calling the denyApprove method of a Mailing did not require the user right 'Approve mailing'. Setting the global visibility of an attribute, on the ohter hand, falsly required the right to access the administration list.

1.6. Inxmail API 1.10.1

Changes in API 1.10.1, since Inxmail Professional 4.3.2

- **New: Support for new "unsubscribe not in list" feature**
  Starting with Inxmail Professional 4.3.2 it is possible to unsubscribe recipients who are no longer member of the list at hand (neither subscribed nor unsubscribed). This feature can be enabled (either global or list specific) using the corresponding list property. Also, there are a couple of new subscription log entry types related to that feature.

Note: New handling of subscription log entry types
Starting with Inxmail Professional API version 1.10.1, the subscription log entry types PENDING_SUBSCRIPTION_DONE, PENDING_UNSUBSCRIPTION_DONE and LIST_UNSUBSCRIBE_HEADER_UNSUBSCRIPTION are no longer converted to VERIFIED_SUBSCRIPTION or VERIFIED_UNSUBSCRIPTION respectively, but are returned as they are. Furthermore, the new NOT_IN_SYSTEM_UNSUBSCRIPTION type marks all unsubscription attempts of recipients who are not registered in Inxmail Professional.

1.7. Inxmail API 1.10.0

Changes in API 1.10.0, since Inxmail Professional 4.3

- **New: Trigger mailing management**
  The new TriggerMailingManager can be used to create, edit, approve, activate, delete and retrieve trigger mailings. This also includes a new command which enables actions to send action mailings.
- **New: Visibility of recipient attributes**
  Two new methods were added to check the visibility of recipient attributes in a list.
- **New: Retrieve recipients by key**
  With the new findByKey(s) and findAllByKey(s) methods it is now very easy to retrieve recipients by their key (e.g. the email address).

Info: New version of .NET framework required
With version 1.10.0 of the Inxmail Professional API, .NET Framework 4.0 is required. If you are not already using the .NET Framework 4.0, be sure to download and install an appropriate SDK. The client profile is not sufficient.

- **Bugfix: selectAll method in FilterManager - All languages**
  A server bug in Inxmail Professional 4.2 caused the selectAll method in the FilterManager to always return an empty result set.
- **Bugfix: Retrieval of command data - .NET**
  Due to a bug it was not possible to retrieve data associated with a command (e.g. SetValueCommand) in the .NET API.
- **Bugfix: Retrieval of SetValueAction command type - All languages**
  Due to a server bug in Inxmail Professional 4.2 the command type of a SetValueAction was returned incorrectly (CMD_TYPE_ABSOLUTE and CMD_TYPE_RELATIVE were swapped).

- **Bugfix: scheduleMailing method in MailingManager - All languages**
  A server bug in Inxmail Professional 4.2 prevented the scheduleMailing method in the MailingManager to throw a SecurityException if the mailing is in the DRAFT state and the API user does not have the right to bypass the approval process.
- **Bugfix: unlock method in MailingManager - All languages**
  The unlock method in the MailingManager did always return false.
- **Bugfix: Import of invalid design collections - All languages**
  Importing a non-ITC file as design collection returned null instead of throwing an appropriate exception.
- **Bugfix: DataException on invalid filters in FilterManager - PHP**
  Due to a bug in the PHP API the commitUpdate method of the Filter class threw a DataException instead of an UpdateException.
- **Bugfix: Error handling in ListManager - PHP**
  A bug in the ListManagerImpl class of the PHP API caused an incorrect behaviour regarding the error handling.
- **Bugfix: Error handling in Mailing - PHP**
  Another bug in the MailingImpl class of the PHP API caused an incorrect behaviour regarding the error handling.
- **Bugfix: Last modification date of filters / target groups - All languages**
  Creating or editing filters (aka target groups) using the API did not change the last modification date.
- **Bugfix: Select methods in BounceManager - All languages**
  The select methods of the BounceManager threw a NullPointerException if a date parameter was null. From now on, if a date parameter is null, that parameter will be ignored.

1.8. Inxmail API 1.9.0

Changes in API 1.9.0, since Inxmail Professional 4.2

- **New: Webpage management**
  The new WebpageManager can be used to retrieve information about JSPs and HTML forms.
- **New: Inbox management**
  The new InboxManager can be used to retrieve messages received via the inbox.
- **New: Visibility of recipient attributes**
  It is now possible to set the visibility of recipient attributes either for a single list or all lists via the AttributeManager.
- **New: Mailing approval deadline / escalation**
  Two new methods were added to the MailingManager to retrieve the approval deadline and escalation dates.
- **New: Design collection display name**
  A new method in the DesignCollection class can be used to retrieve the display name of the design collection.
- **New: Subscription management**
  The SubscriptionManager can now be used to update recipient attributes during unsubscription and is able to handle mailing references.

- **New: Multiple defined header fields**
  Using a new method in the MailContent class it is now possible to retrieve multiple defined header fields.

Note: Incorrect documentation regarding close() methods
The documentation of some classes stated that the close() method would also be called when the corresponding object is garbage collected. This is not correct and can lead to memory problems, especially when applied to sessions. It is strongly recommended to close all sessions and row sets manually. The documentation was corrected accordingly.

Note: New handling of orphaned unsubscriptions
Starting with Inxmail Professional 4.2, unsubscriptions of recipients who are no longer member of the list at hand (neither subscribed nor unsubscribed) will be marked as NOT_IN_LIST_-UNSUBSCRIPTION.

- **Note: Documentation improvement for Java and PHP**
  The Java and PHP documentation was completely revised to be more comprehensive and understandable. Also, the PHP documentation now takes into account some PHP specifics.

- **Note: Report documentation error**
  There was an error in the documentation of the ClickReactionTimeResponse report which was corrected.

Bugfix: Error handling in DesignCollectionManager - All languages The importDesignCollection method threw a NullPointerException if the template feature was not available. From now on a FeatureNotAvailableException will be thrown. Therefore, existing code relying on a NullPointerException being thrown must be changed to catch the FeatureNotAvailableException instead.

- **Bugfix: existsTestRecipient in Utilities class - Java**
  The existsTestRecipient method in the Utilites class threw a NullPointerException in version 1.8.0 of the Java API. The PHP5 and .NET APIs are not affected.

- **Bugfix: Scheduling of mailings - PHP5**
  A bug in the Mailing class made it impossible to schedule a mailing properly.

- **Bugfix: Approval of mailings - PHP5**
  Due to a bug in the Mailing class it was not possible to request the approval of a mailing.

- **Bugfix: setAttributeValue in RecipientContext - PHP5**
  The setAttributeValue method in the RecipientContext class threw a fatal error.

- **Bugfix: Select methods in BounceManager - PHP5**
  The selectBefore() and selectAllBounces() methods in the BounceManager class threw errors.

- **Bugfix: Test recipient management - PHP5**
  The creation and manipulation of test recipients, as well as the retrieval of test recipient attributes was not possible.

- **Bugfix: Subscription log - PHP5**
  The retrieval of log entries using the SubscriptionManager and the retrieval of the log message using the SubscriptionLogEntryRowSet did not work.

- **Bugfix: Resubscription of recipients - PHP5**
  Unsubscribed recipients could not be resubscribed using the UnsubscriptionRecipientRowSet.

- **Bugfix: formatAttributeChoice in FormatChoicePropertyFormatter - PHP5**
  The formatAttributeChoice method in the FormatChoicePropertyFormatter class threw a fatal error.

- **Bugfix: parseApprovalPropertyValue in PropertyFormatter - PHP5**
  The parseApprovalPropertyValue method in the PropertyFormatter class threw a fatal error.
- **Bugfix: Plugin store - PHP5**
  The get and put methods in the PluginStore did not work properly. The get method threw an error, the put method simply blocked and did nothing.
- **Bugfix: Who am I - PHP5**
  A bug prevented the whoAmI method in the UserContext from returning the user object.
- **Bugfix: Exception handling - PHP5**
  Some exceptions could not be rebuild correctly and threw warnings instead.

1.9. Inxmail API 1.8.0

Changes in API 1.8.0, since Inxmail Professional 4.1

- **New: Linktypes for new (un)subscription links**
  DataAccess is now extended with new link types.
- **New: Subscription log extended with pending states**
  Pending un- and subscription states added.

Bugfix: Heartbeat problem in Java api 1.7.2
In Inxmail Professional API 1.7.2 for Java the heartbeat is not started correct, so we recommend to switch to this version or 1.7.1

- **Bugfix: remove() in BOResultSet of MailingTemplates does not work**
  The remove() method doesn't work in versions before 1.8.0, the mailing templates are not deleted when calling boresultset.remove(new Inde...);.
- **Internal Changes: Changing build to Maven**
  As preparation of an Inxmail hosted Maven repository switched build from Ant to Maven.

1.10. Inxmail API 1.7.2

Changes in API 1.7.2, since Inxmail Professional 4.0.2

- **New: Clone mailing**
  Now it is possible to clone a mailing with a single method call (MailingManager.cloneMailing(...)).
- **New: Hessian uses GZIP compression**
  When using the Hessian protocol as default GZIP compression is now activated.

1.11. Inxmail API 1.7.1

Changes in API 1.7.1, since Inxmail Professional 4.0.1

- **New: Mailing creation date**
  The creation date of a mailing is now available.

1.12. Inxmail API 1.7.0

Changes in API 1.7.0, since Inxmail Professional 4.0.0

- **New: Plugin data store**
  Information can now be stored on the Inxmail Professional System.
- **New: Plugin whoami**
  Plugins can now ask the Inxmail Professional Server which user currently is using the plugin.
- **New: Feature id**
  Introducing a new feature id for the template agent.

Bugfix: Fixed bug in Hessian
There is a major bug in the implementation of the Hessian protocol, so we recommend to switch to this api version.

1.13. Inxmail API 1.6.2

Changes in API 1.6.2, since Inxmail Professional 3.8.2.16

- **New: DataAccess extended**
  Click data can be searched by time.
- **Bugfix: Fixed bug in .NET**
  It was not possible to login in shorten times.

1.14. Inxmail API 1.6.1

Changes in API 1.6.1, since Inxmail Professional 3.8.2

- **New: Testmailing can be send with test profile**
  Test mailing can be send with test profile.
- **New: Bounce handling extended**
  Bounce handling is extend, now it is possible to fetch recipient attributes.
- **New: Inxmail Professional ASP Portal**
  The Java and .NET API has added support for the new Inxmail Professional ASP Portal. For using the PHP5 API please read chapter 3.1.
- **Bugfix: Fixed minor bug in PHP5 API**
  Constant has a wrong name

1.15. Inxmail API 1.6.0

Changes in API 1.6.0, since Inxmail Professional 3.8.1

- **New: Testprofiles can be used**
  It is possible to create/delete/change testprofiles. Also creating a preview of mailing with test recipients is possible.
- **New: Hardbounce attribute**
  A hardbounce attribute is introduced with this version of the Inxmail Professional API.
- **New: Unsubscribed recipients can be retrieved**
  Now it is possible to access the unsubscribed recipients of a list. Also it possible to resubscribe them and unsubscribe recipients.
- **New: Approval for mailings can be used**
  Approval methods are added to the Inxmail Professional API. Also methods for activating approval for a list are added.

- **New: Approver management**
  It is possible to create/delete/change approver.
- **New: Multiple target groups in mailings**
  Multiple target groups can be set for mailings.
- **New: Actions added**
  New actions for handling subscription and unsubscription introduced.
- **New: Security check for Plugin access**
  Plugins should use the new login method with plugin secret id. Also recipient attributes supports access rights, so only allowed attributes can be access. For more information please read the Plugin documentation.
- **Bugfix: Fixed minor bug in PHP5 API**
  Removing warning when unset values are accessed.
- **Bugfix: Missing subscription log entries added**
  Adding double opt-in/out log entries to the subscription log entries.
- **Bugfix: Hessian protocol serializer does not work correct in .NET API**
  Not the correct serializer was used to serialize arrays, so many error messages shown in the server logs. But the API works as expected.

1.16. Inxmail API 1.5.0

Changes in API 1.5.0, since Inxmail Professional 3.8.0

- **New: Login with token possible**
  It is possible to login with a token which is created by the Inxmail Professional Client. This can be used for plugin development.
- **New: Buildmode**
  Adding a new buildmode for building mailings with simple links.
- **Bugfix: Fixed bug in PHP5 API**
  In the Inx_Api_Recipient_RecipientRowSet was an error when updating date values and fetching recipients backwards.
- **Bugfix: Fixed bug in PHP5 API**
  In the Inx_Api_Recipient_BatchChannel was an error when adding recipients.
- **Bugfix: Fixed bug in PHP5 API**
  Fixed selecting bounces by mailing id.
- **Bugfix: Fixed bug in PHP5 API**
  In the Inx_Api_Recipient_BatchChannel was an error when adding recipients.
- **Bugfix: Fixed bug in PHP5 API**
  Fixed error when retrieving mailings.
- **Bugfix: Fixed bug in .NET and Java API**
  Ignoring case considerations when getting user attribute.
- **Bugfix: Fixed several bugs in .NET**
  Fixed timeout problems when using Hessian and recreating sessions with Hessian.

- **Bugfix: Default API role have access to the recipient data**
  The default API user role can only login over the API and nothing more. Added missing check in recipient context.

- **Bugfix: Direct bounces have no sender address**
  Fixed problem when retrieving bounces with have no sender address.

1.17. Inxmail API 1.4.4

Changes in API 1.4.4, since Inxmail Professional 3.7.1

- **New: Access subscription log**
  Retrieving of the subscription log entries is now possible.
- **New: Set/Get of a mailing name**
  Mailings which are created over the api can set a mailing name.
- **New: Getting server time**
  With this version of the Inxmail API it is possible to get the time of the Inxmail Professional Server.

Note: Inxmail API for .NET supports .NET Framework >= 2.0
Since this version only .NET 2.0 and higher is supported.

- **New: .NET supports now Hessian**
  When using the .NET API, it is now possible to use the Hessian protocol.
- **New: .NET has ”Strong Name” for using GAC**
  Adding a ”Strong Name” to the .NET API which makes it possible to store the assembly in the global assembly cache.
- **New: .NET is now COM Visible**
  Now the API can be used as COM objects in programming languages like Delphi or FoxPro.
- **Bugfix: Fixed bug in PHP5 API**
  In the Inx_Api_Recipient_RecipientRowSet was an error when updating boolean values.

1.18. Inxmail API 1.4.3

Changes in API 1.4.3, since Inxmail Professional 3.7

- **New: Search for link names in Data Access**
  Now it is possible so search for link data with the name of link.
- **New: Getting list size**
  Adding a new method for getting the list size and the list size computation date.
- **New: Getting more info of a sent mailing**
  Introducing a new object which contains infos about the sent mailing, such as average mail size or number of bounces
- **New: Bounce handling**
  Adding a new service for retrieving bounce mails. With Inxmail Professional 3.7 you can get which recipient has bounced.
- **New: Search for blacklist entries**
  Adding new methods for searching in blacklist with given search dates.
- **New: Secure login available for PHP5 and .Net API**
  Now secure login in all three programming languages available.

New: Property for test recipient deprecated
In Inxmail Professional 3.7 the test recipient in a list is replaced by test profiles. The test recipient property will be removed in further versions of the Inxmail API. It should not be used anymore.

1.19. Inxmail API 1.4.2

Changes in API 1.4.2

- **New: Additional temporary mailing method**
  This new methods makes easier to send a temporary mailing without using a recipient id.
- **New: Data Access has a new method**
  Adding a new method for retrieving link data which uses the new link type opening rate.
- **New: More Login Exceptions**
  Adding new Login Exception for the new password behavior, for example password timed out.

New: Bugfix in Hessian API
Customers which use Hessian protocol are strongly recommended to change to the new version, which is in the API-Zipfile. There was a bug in transfering boolean values between client and server.

1.20. Inxmail API 1.4.1

Changes in API 1.4.1

- **New: Buildmode for Mailings**
  Added two new modes for building Mailings. See Chapter 3.9.2 "Mail Preview".

1.21. Inxmail API 1.4.0

Changes in API 1.4.0

- **New: Hessian Protocol**
  Added support of a faster protocol for Java.
- **New: Textmodule management**
  Added management of textmodules, allowing to add, select, and change textmodules via API.
- **New: Mailing template management**
  Added management of mailing templates, allowing to add, select, and change mailing templates via API.
- **New: Design collection management**
  Added management of design collections, allowing to add, select, and change design collections via API.

1.22. Inxmail API 1.2.0

Changes in API 1.2.0, since Inxmail Professional 3.2 build 060130.

- **New: Actions management**
  Added management of actions, allowing to add, select, and change actions via API.
- **New: Filter for "MailingManager.select"**
  Introduced new filter options to select Mailings. See Chapter 3.9.2 "Retrieval of Mailings".

- **New: createRecipient with alternative key attribute**
  Introduced option for Batch Channel to operate with alternative key attribute instead of email address. See chapter 3.6.7.
- **New: Blacklist management**
  Added management of blacklist, so blacklist rules can selcted, deleted, added and changed. See chapter 3.9.13, "BlacklistManager".
- **Doc: Batch Channel**
  Added missing documentation of return value from `executeBatch`: Values above zero are recipient ids.

2. Introduction

The *Inxmail API* (Application Programming Interface) enables other applications to access and control Inxmail Professional and Enterprise. Thus, third parties can extend Inxmail by adding own functionality and services. It is shipped as an integral part of Inxmail. No license is needed for the local, anonymous login. Otherwise an "API Module License" must be acquired.
The technologies used for the API are independant of platforms and programming languages. Therefore, API calls can be done from software written in any programming language, running on any platform, like Java applications on Linux, or .NET apps on Windows.
Remote API calls are transported over HTTP/HTTPS. The API is based on SOAP (Simple Object Access Protocol), which itself utilizes XML. To ease writing software with the API, default "wrappers" for Java, .NET and PHP are provided, called "Inxmail API for Java", "Inxmail API for .NET", and "Inxmail API for PHP". Please note that direct access to the SOAP layer or other wrappers are not supported by Inxmail GmbH.

2.1. Security Issues

The API differentiates between local and remote calls. Local calls do not need a username or password to log in, and can only be performed from Java applications which are running on the same computer and inside the same virtual machine as the Inxmail server.
Remote calls can be performed from any computer having access to the Inxmail server via HTTP. Enabling remote API calls might pose a security threat. Therefore, these calls need to login with a username and password, and the target user needs to have the user right *"Remote API login"* enabled. To secure remote calls even further, the *"Allowed IP mask"* in the user's definition can disallow logins which do not match this mask.
User credentials on login are communicated to the server either in plain text or encrypted using challenge response encryption with SHA256. Since encryption is a time consuming process, many developers opt for plain text over SSL-secured HTTP (HTTPS).

2.2. System Requirements

To use the API, access to an Inxmail Server is necessary. For remote calls, the server calling Inxmail needs to be able to access the Inxmail Server via HTTP.
Inxmail PHP API was previously developed to run under PHP 5.
The execution time of scripts may be longer as the default limit. To extend the allowed execution time, change the php.ini file or add the following statement in your php-file "set_time_limit(3600);"
Remote calls to the API need a valid API license on the Inxmail Server.

2.3. Inxmail API for PHP

2.3.1. Inxmail API for PHP4

The developer of PHP have anounced that PHP4 discontinued after the 2007-12-31 and only some security fixes available till the 2008-08-08.
So we decided to migrate the Inxmail API for PHP4 to PHP5. After the release of Inxmail API for PHP5, major bugs and security issues will be fixes in the PHP4 API until August 2008, but PHP4 API will not see new development efforts. Of course, Inxmail API for PHP4 can still be used. However, we recommend all API developers to migrate their PHP4 API applications to the new Inxmail

API for PHP5.

The samples in this document are for the Inxmail API for PHP5.

2.3.2. Inxmail API for PHP5

Inxmail PHP API library is in the inxmail_api directory.

Note: The internal soap library is used for the communication to the Inxmail Professional Server. So make sure that soap is installed and activated in your PHP5 installation.

To start up, read the samples you find in the samples directory. To run them, you can use the run.php or run.bat.
For using inxmail interface, first registration of autoloader is needed.

```
require_once './inxmail_api/Apiimpl/Loader.php';
Inx_Apiimpl_Loader::registerAutoload();
```

Naming conventions

For PHP5 inxmail interface naming conventions are borrowed from Zend Framework and PEAR. The class names contain information about directory in which the file is (package), e. g. class Inx_Api_Action_ActionManager is in a directory called "Action" which is in directory called "Api". "Inx" is a prefix to indicate, that this is part of the Inxmail interface.

3. API Description

Following chapters give detailed information about how to program using the Inxmail API with the PHP. Code examples are written for PHP software.

3.1. Sessions

All API calls need a valid session on the Inxmail Server. The *Inx_Session class* is used to establish connections to the Inxmail Server and is the starting point for all applications using the API.

3.1.1. Login and Logout

Remote Named Sessions

Remote logins can be performed from any computer which have access to the Inxmail Server via HTTP[1]. Enabling remote API calls might pose a security threat. Therefore, these calls need to login with a username and password, and the target user needs to have the user right *"Remote API login"* enabled. To secure remote calls even further, the "Allowed IP mask" in the user's definition can disallow logins which do not match this mask. For remote calls, username and password have to be always available. The target user needs to have the user right "Remote API login" enabled:

```
require_once './inxmail_api/Apiimpl/Loader.php';
Inx_Apiimpl_Loader::registerAutoload();
$session = Inx_Api_Session::createRemoteSession("http://127.0.0.1/inxmail0" ,
                "api-user" , "test");
```

Full example:

```
<?php
  require_once './inxmail_api/Apiimpl/Loader.php';
  Inx_Apiimpl_Loader::registerAutoload();

  $inx_server = "http://127.0.0.1/inxmail0";
  $inx_user = "api-user";
  $inx_pass = "test";

  try {
      $oSession = Inx_Api_Session::createRemoteSession (
      $inx_server, $inx_user, $inx_pass ) ;
  }
  catch ( Inx_Api_LoginException $x ) {
      echo $x->getTraceAsString();

  }
  catch ( Exception $x ) {
      echo $x->getTraceAsString();
  }
  $oSession->close();
?>
```

User credentials on login are communicated to the server either in plain text or encrypted using a challenge response method with SHA256 encryption. Since encryption is a time consuming process, many developers opt for plain text over SSL-secured HTTP (HTTPS). Encryption is enabled with the extract parameter to the `createRemoteSession` method:

[1] Remote login requires an API license on the Inxmail Server!

```
$session = Inx_Api_Session::createRemoteSession(
    "http://127.0.0.1/inxmail0", "api-user",
    "test", true );
```

3.1.2. Using Proxy Servers

If you need to pass through a Proxy server, set the Proxy parameters before creating the session:

```
Inx_Api_Session::setProperty('http.proxyHost', '127.0.0.1');
Inx_Api_Session::setProperty('http.proxyPort', 8118);
Inx_Api_Session::setProperty('http.proxyUser', 'proxyuser');
Inx_Api_Session::setProperty('http.proxyPassword', 'proxypassword');
```

If you require SSL to connect to the proxy and should encounter a 'Bad Request' error message, explicitly enabling the SNI (Server Name Identification) feature with the following option might help:

```
Inx_Api_Session::setProperty('http.enableSNI', 'true');
```

3.2. Getting the Inxmail Professional Server time

The server time is needed if the Inxmail Professional Server is in another timezone located as the programm which uses the Inxmail API. The GMT and daylight saving time offset is given in milliseconds.

```
$st = $s->getServerTime();
print( $st->getDatetime() . " " . $st->getGMTOffset() .
                            " " . $st->getDSTOffset() . " "
                            . $st->getTimezoneId() );
```

3.3. Sending temporary Mails

The Inxmail API provides a mechanism for sending temporary mails to a single recipient. The advantage of this mechanism is that the recipient must not in the Inxmail System or subscribed in a list. These mails are not personalized, not trackable and not saved in Inxmail.

```
$listContextManager = $oSession->getListContextManager();
$list = $listContextManager->findByName(Inx_Api_List_SystemListContext::NAME);
$tempSender = $oSession->getTemporaryMailSender();

$tempMailing = $tempSender->createTemporaryMail($list);
$tempMailing->updateRecipientAddress("recipient@test.invalid");
$tempMailing->updateSenderAddress("sender@test.invalid");
$tempMailing->updateReplyToAddress("replyto@test.invalid");
$tempMailing->updateSubject("Temporary Mailing");

$tempMailing->setContentHandler('Inx_Api_Mailing_HtmlTextContentHandler');
$oContentHandler = $tempMailing->getContentHandler();

$oContentHandler->updateContent('<html><head></head><body>Hi there, <br>
        this is a temporary mailing</body></html>');
$blSuccess = $tempSender->sendTemporaryMail($tempMailing);
if ($blSuccess) {
  echo 'Mailing succeeded';
}
else {
  echo 'Mailing failed';
}
```

3.4. Inx_Api_BusinessObjects and Inx_Api_BOResultSets

The API gives access to objects of Inxmail, which are called "BusinessObjects". For example, a *mailing lists* in Inxmail is such a Business Object.

Values of Inx_Api_BusinessObjects and Inx_Api_BOResultSets can be changed with the "update" methods (like "updateName"). By calling "commitUpdate" on such an object, changes will be passed to the server. Rollback is done by the "reload" method, which reloads the object and discards all uncomitted changes.

A *Inx_Api_BOResultSet* is a list of BusinessObjects. The result set can be used to browse through this list, and to remove elements of the list.

From Inxmail Professional API 1.11.4, Inx_Api_BOResultSet implements Iterator. This enables you to use a for-each loop on the result set. The following sample demonstrates the different ways to retrieve business objects from the result set.

```
$oMailingManager = $oSession->getMailingManager();
$oBOResultSet = $oMailingManager->select($oListContext, Inx_Api_Mailing_MailingManager::STATE_FILTER_ALL);

// traditional way of iterating through the result set (still works)
for( $i = 0; $i < $oBOResultSet->size(); $i++ )
{
    $mailing = $oBOResultSet->get($i);
    echo $mailing->getName() . '<br>';
}

// iterate through the result set using for-each loop
foreach( $oBOResultSet as $mailing)
{
    echo $mailing->getName() . '<br>';
}

$oBOResultSet->close();
```

The interfaces of Inx_Api_BusinessObjects and Inx_Api_BOResultSets are defined as follows:

```
interface Inx_Api_BusinessObject
  public function getId();
  public function commitUpdate();
  public function reload();

interface Inx_Api_BOResultSet extends Iterator
  public function get( $iIndex );
  public function size();
  public function remove( Inx_Api_IndexSelection $oSelection );
  public function close();
```

The recipient addresses are one of the exceptions of this rule. They are managed not in Inx_Api_BOResultSets, but by the specialized class "Inx_Api_Recipient_RecipientRowSet".

3.5. Inx_Api_List_ListContext Management

A *Inx_ListContext* corresponds to a folder in Inxmail, like a mailing list or the system folder. The *Inx_ListContextManager* is used to access and manipulate these folders.
The Inx_Api_List_ListContext is an interface with following concrete implementations:

Inx_Api_List_AdminListContext : The "Administration" list.

Inx_Api_List_FilterListContext : This is a "Dynamic Mailing List" in Inxmail. It has methods for getting and setting the filter condition the dynamic list is based on.

Inx_Api_List_StandardListContext : A normal mailing list.

Inx_Api_List_SystemListContext : The "System list" in Inxmail.

To browse though all accessible lists (corresponding to the user's access rights), a result set can be optained by calling *selectAll* of the Inx_Api_List_ListContextManager.

```
$listContextManager = $oSession->getListContextManager();
$oBOResultSet = $listContextManager->selectAll();
for ($i=0; $i < $oBOResultSet->size(); ++$i) {
        $l = $oBOResultSet->get($i);
        echo "List-Id: " . $l->getId() . "\n";
        echo "Name: " . $l->getName() . "\n";
        echo "Description" . $l->getDescription(). "\n";
}
$oBOResultSet->close();
```

3.5.1. Creating, Searching and Naming Lists

New mailing lists are created by the Inx_Api_List_ListContextManager, which can be optained from the session object. The list will not be created until `commitUpdate` has been called.

```
$listContext = $oSession->getListContextManager()->createStandardList();
```

Set the list name with the `updateName` method. If the list cannot be renamed (for example, a list with that name already exists), an `Inx_Api_UpdateException` will be returned.

```
$listContext->updateName( $name );
$listContext->commitUpdate();
```

The Inx_Api_List_ListContextManager can be consulted to find lists by their name:

```
$listContextManager = $oSession->getListContextManager();
$listContext = $listContextManager->findByName( $name );
```

For `Inx_Api_List_FilterListContext`, a filter condition can be defined.

```
$oFilterListContext = $oSession->getListContextManager()->createFilterList();
$oFilterListContext->updateName("NewYorkList");
$oFilterListContext->updateFilterStmt(" City = \"New York\"");
$oFilterListContext->commitUpdate();
```

3.5.2. Size of Lists

With the Inxmail API 1.4.3 you are able to get the list size and the computation date of the list size. It is stored in the Inx_Api_List_ListSize object which can be retrieved by using the following methods in the Inx_Api_List_ListContext:

```
public function getListSize();
public function getListSize( $computeNow );
```

Caution: Refreshing the list size can produce a very high load on the Inxmail server. Use this with caution.

3.5.3. List properties

Mailing lists have properties, which control behaviour like the maximal sending performance or which are used by features.
The properties can be accessed through these methods:

```
public function findProperty( $sPropertyName );
public function selectProperties();
```

The property class can be found in inxmail_api/Api/Property/Property.php.

Note: The property for test recipient is deprecated. Do not use it anymore, it will be removed in further versions.

3.6. Inx_Api_Recipient_RecipientContext

The *Inx_Api_Recipient_RecipientContext* is used to access recipient data. Getting this context from the session will get a snapshot of the current attribute defined. This snapshot will be used for the lifetime of the context, changes in the underlying attribute configuration won't be reflected to it. This ensures that you can savely work with recipient data, even if other users possibly add or change attributes.
Following example illustrates how to list the email addresses of all recipients in a list named "test":

```
$oRecipientContext = $oSession->createRecipientContext();
$oRecipientMetaData = $oRecipientContext->getMetaData( ) ;
$oAttrEmail = $oRecipientMetaData->getEmailAttribute();
$oListCManager = $oSession->getListContextManager();
$oList = $oListCManager->findByName( 'test' );
$oRecipientRowSet = $oRecipientContext->select($oList, null, null,
                                    $oAttrEmail, Inx_Api_Order::ASC);

while ($oRecipientRowSet->next())
    echo $oRecipientRowSet->getString($oAttrEmail). "\n";
$oRecipientRowSet->close();
```

3.6.1. Adding New Recipients

In the Inxmail client, the table of recipients has an "insert" row, which is always the last in a table and marked with an asterisk. Adding recipients in the API is like in the Inxmail client: move to this "insert" row, edit the email address and then all the other data fields. Remember to commit your changes, otherwise they don't be reflected on the server. Following code will fails if the address is already in the system.

```
$oRecipientContext = $oSession->createRecipientContext();
$oRecipientMetaData = $oRecipientContext->getMetaData();
$oRecipientRowSet = $oRecipientContext->createRowSet() ;
// Move to the "insert" row and set the values :
$oRecipientRowSet->moveToInsertRow ( ) ;
$oRecipientRowSet->updateString ( $oRecipientMetaData->getEmailAttribute(),
                                    "andi@company.com" ) ;
$oRecipientRowSet->updateString ( $oRecipientMetaData->getUserAttribute( "Firstname" ),
                                    "Andi" ) ;
try
{
    $oRecipientRowSet->commitRowUpdate();
}
catch ( Inx_Api_Recipient_DuplicateKeyException $x )
{
// A recipient with the specified?mail address is already present
}
$oRecipientRowSet->close() ;
$oRecipientContext->close();
```

Adding more than one recipients at a time is very slow. For large amount of data use a *Inx_Api_Recipient_BatchChannel*.

3.6.2. Inx_Api_Recipient_BatchChannel

The `createRecipient` and `selectRecipient` methods are used to create and/or select a recipient. After creating or selecting a recipient, the following batch commands operate on this until another recipient is selected.

```
public function createRecipient( $sKeyValue, $blSelectIfExistant );
public function selectRecipient( $sKeyValue );
```

Following example shows how to add two new addresses and change their "Firstname" attribute. If the addresses exist already, this value will be overwritten.

```
$oRecipientContext = $oSession->createRecipientContext();
$bc = $oRecipientContext->createBatchChannel();
$oRecipientMetaData = $oRecipientContext->getMetaData() ;

$bc->createRecipient ( "mueller@yourcompany.com" , true ) ;
$bc->write($oRecipientMetaData->getUserAttribute( "Firstname" ) , "George" ) ;
$bc->createRecipient ( "clinton@yourcompany.com", true ) ;
$bc->write($oRecipientMetaData->getUserAttribute( "Firstname" ) , "Bill" ) ;

$ret= $bc->executeBatch();
```

Each command to the *Inx_Api_Recipient_BatchChannel* results in a value in the returned integer array. By scanning the array, you can find out which of the commands have been executed, and which have not. The integers are of these constants:

`RESULT_COMMITED` - The batch command has been committed.

`RESULT_NOT_COMMITTED` - The command has not been comitted.

`RESULT_FAILURE_ILLEGAL_VALUE` - The command has not been executed because the value was not allowed.

`RESULT_FAILURE_BLOCKED_BY_BLACKLIST` - The email address cannot be inserted or updated, since it is blocked by the blacklist.

`RESULT_FAILURE_DUPLICATE_KEY` - The email address cannot be inserted or updated since it already exists.

`RESULT_FAILURE_KEY_NOT_FOUND` - The unique key was not found by the `selectRecipient()` method.

`values above zero` - Recipient id

3.6.3. Searching Recipients

To search recipients, pass a filter condition to the select method of the *Inx_Api_Recipient_RecipientContext*. You can also use the Inxmail Professional Functions which are documented in the Inxmail Professional Client manual.

```
$oRecipientContext = $oSession->createRecipientContext();
$oRecipientMetaData = $oRecipientContext->getMetaData () ;
$oSortAttr = $oRecipientMetaData->getEmailAttribute() ;
$sFilter = "email LIKE \"%yourcompany.com\"" ;
$oRecipientRowSet = $oRecipientContext->select(null, null,
      $sFilter, $oSortAttr, Inx_Api_Order::ASC);
```

Since version 1.10.0 of the Inxmail Professional API, there is an easier way to accomplish this task. If you wish to retrieve a recipient with a specific key (e.g. the email address), you can use the following snippet:

```
$oRc = $oSession->createRecipientContext();
$oResult = $oRc->findByKey( 'recipient.of@yourcompany.invalid' );
$oResult->next();
```

In some environments the recipient key may be ambiguous. To retrieve all recipients with the given key, use the `findAllByKey` method instead. It is also possible to retrieve the recipients for a list of keys. To accomplish this task, use the `findByKeys` or `findAllByKeys` method.

Following code demonstrates how to search a recipient with its identifier:

```
$sFilter = "RecipientId() = 1234";
$oRecipientRowSet = $oRecipientContext->select( null, null,
                $sFilter , $oSortAttr , Inx_Api_Order::ASC );
$oRecipientRowSet->next();
```

3.6.4. Controlling List Membership

List membership is controlled by a "subscription date" value, which exists for each standard mailing list. To add a recipient to a list, update this value with a date. Remove a recipient by setting this value to `null`:

```
$oRecipientContext = $oSession->createRecipientContext();
$oRecipientMetaData = $oRecipientContext->getMetaData();
$oList = ...get list context...
$oRecipientRowSet = ...find recipient...
// Add recipient to list:
$oRecipientRowSet->updateDatetime( $oRecipientMetaData->getSubscriptionAttribute( $oList )
                                   , date('c') );
// Remove recipient from list:
$oRecipientRowSet->updateDatetime( $oRecipientMetaData->getSubscriptionAttribute( $oList )
                                   , null );
```

3.6.5. Deleting Recipients

Deleting a recipient from the system is done by calling `deleteRow` or `deleteRows` on the Inx_Api_Recipient_RecipientRowSet:

```
$oRecipientRowSet = ...find recipient...
// Delete current row :
$oRecipientRowSet->deleteRow();
// Delete multiple recipients :
$oRecipientRowSet->deleteRows ( new Inx_Api_IndexSelection ( 0 ,10) );
```

3.6.6. Updating Recipients

There are two ways to update values of a recipient. If there is only one recipient to be changed, the following sample code demonstrates updating a single recipient.

Note: For changing the hardbounce attribute the api user needs the recipient changing right.

```
$oRecipientRowSet = ...find recipient...
// Update value :
$oRecipientRowSet->updateBoolean($oMetaData->getUserAttribute("promotion"), true);
$oRecipientRowSet->commitRowUpdate();
```

For more than a few recipients it is better to let the server do the work, as walk through the result set and change every recipient. This is already faster than walking through the result set.

```
$oRecipientRowSet = ...find recipients . . .
// Update all found recipients
$oRecipientRowSet->setAttributeValue( $oMetaData->getUserAttribute( "promotion" )
                                      , true ) ;
```

3.6.7. Using alternative key instead of email address

In most use cases, the email address is used as key attribute for recipient management. However, in some cases alternative key attributes are needed, e.g. an "account number".
Therefore, the Batch Channel offers the possibility to select text or integer values as key instead of the email address. Of course, the email address remains unique if the "dublicates allowed" option of Inxmail database is not set.
Following method creates a Batch Channel with an alternative key attribute to select the recipients. Allowed data types of the key attribute are either `Inx_Api_Recipient_Attribute::DATA_TYPE_STRING` or `Inx_Api_Recipient_Attribute::DATA_TYPE_INTEGER`. If the key attribute (e.g. due to manual data import), it will not be determined which one of these will be selected by the Batch Channel methods.

```
public function createBatchChannel( Inx_Api_Recipient_Attribute $oSelectAttribute );
```

Example: Change email address of client with account number "206.914.112"

```
$oRecipientContext = $oSession->createRecipientContext();
$oAccountId = $oRecipientContext->getMetaData()->getUserAttribute("AccountID");
$oBatchChannel = $oRecipientContext->createBatchChannel ($oAccountId) ;
// Select customer with Account-ID "206.914.112"
$oBatchChannel->selectRecipient("206.914.112");
$oBatchChannel->write( $oRecipientContext->getMetaData()->getEmailAttribute(),
                       "new@email.invalid") ;
$aRet = $oBatchChannel->executeBatch();
```

3.6.8. Unsubscribed recipients

Since Inxmail Professional 3.8 unsubscribed recipients are shown in a special table. Since Inxmail Professional API 1.6.0 these unsubscribed recipients can be accessed by the Inxmail Professional API. The
Inx_Api_Recipient_RecipientContext contains the following methods to retrieve a
Inx_Api_Recipient_UnsubscriptionRecipientRowSet which contains the unsubscribed recipients.

```
public function selectUnsubscriber( Inx_Api_List_ListContext $list, Inx_Api_Filter_Filter $oFilter=null,
        $sAdditionalFilter=null, Inx_Api_Recipient_Attribute $oOrderAttribute=null, $iOrderType=null );
```

3.7. AttributeManager

Using the *Inx_Api_Recipient_AttributeManager*, attributes (columns) can be manipulated. Following example illustrates how to create a new text attribute with length of 50 characters:

```
$oSession->getAttributeManager()->create (
  "Firstname",
  Inx_Api_Recipient_Attribute::DATA_TYPE_STRING, 50);
```

Renaming attributes is performed using the `rename` method, removing by calling `remove` in the AttributeManager.
The following example shows how to check the visibility of a few attributes. If the last modification attribute is not visible in the list, it will be made visible. The opposite is true for the subscription attribute. If the lastname attribute is not visible, it will be made visible in all lists:

```
$oRecipientContext = $oSession->createRecipientContext();
$oRecipientMetaData = $oRecipientContext->getMetaData();
$lastModification = $oRecipientMetaData->getLastModificationAttribute();
$subscription = $oRecipientMetaData->getSubscriptionAttribute( $oListContext );
$lastname = $oRecipientMetaData->getUserAttribute( "Lastname" );
$oAttributes = array( $lastModification, $subscription, $lastname );

$oAttributeManager = $oSession->getAttributeManager();
$aVisibility = $oAttributeManager->areAttributesVisibleInList( $oAttributes, $oListContext->getId() );

if( !$aVisibility[$lastModification->getId()] )
  $oAttributeManager->setAttributeListVisibility( $lastModification, $oListContext->getId(), true );

if( $aVisibility[$subscription->getId()] )
  $oAttributeManager->setAttributeListVisibility( $subscription, $oListContext->getId(), false );

if( !$aVisibility[$lastname->getId()] )
  $oAttributeManager->setGlobalAttributeVisibility( $lastname, true );
```

3.8. ApproverManager

The *Inx_Api_Approval_ApproverManager* is used for selecting/removing/creating approvers in Inxmail Professional. The following sample creates a new approver.

```
$lc = ...;
$apm = $session->getApproverManager();
$newApr = $apm->createApprover();
$newApr->updateComment("API created approver");
$newApr->updateEmail("approver@inv.invalid");
$newApr->updateLists( array( { $lc->getId() }));
$newApr->updateName("Approver 1");
$newApr->commitUpdate();
```

3.9. Features

Agents, like "Mailing" or "Subscriptions" are called "Features" in the API language. Which features are available can be optained from the `Inx_Api_Features` interface.
Features are enabled and disabled from the *Inx_Api_List_ListContext*, as following example demonstrates, which enables the "Subscriptions" agent in the chosen mailing list:

```
$oListContext = ...get list context...
$oListContext->enableFeature(Inx_Api_Features::SUBSCRIPTION_FEATURE_ID);
```

Not every feature is accessible for every type of list. For example, "Subscription" feature is available in standard lists, only. The "Mailing" feature can be used in standard and filter lists. If a feature is not available for a list, an `Inx_Api_FeatureNotAvailableException` will be returned.
Features are controlled by their respective managers. As such, there is a "Inx_Api_Mailing_MailingManager" and a "Inx_Api_Subscription_SubscriptionManager".

3.9.1. Inx_Api_Subscription_SubscriptionManager

If the subscription feature is enabled for a standard list, the `Inx_Api_Subscription_SubscriptionManager` can be used to subscribe and unsubscribe recipients. The behaviour is the same as if a recipient subscribes to a list via a web frontent. For example, if **double opt in** is configured, calling `subscribe` will start the normal double opt in subscription process.

```
$oSubscriptionManager = $oSession->getSubscriptionManager();
$attrs = array();
$attrs['Firstname'] = "Max";
$attrs['Lastname'] = "Mustermann";
$iResult = $oSubscriptionManager->processSubscription("Sourceidentifier", "127.0.0.1",
                                $oStandardListContext,"max.musterman@inxmail.de", $attrs );
```

The result is either
`Inx_Api_Subscription_SubscriptionManager::PROCESS_ACTIVATION_SUCCESSFULLY` if the subscription or unsubscripton succeeded, or
`Inx_Api_Subscription_SubscriptionManager::PROCESS_ACTIVATION_FAILED_ADDRESS_ILLEGAL` if the address is not conform to the RFC standard.
Also can be the
`Inx_Api_Subscription_SubscriptionManager` used for retrieving the subscription log entries. The following methods can be used for getting the subscription log entries. Each methods returns an rowset which contains the entries.

```
function getAllLogEntries( $rc, $attrs );
function getLogEntriesForList( $lc, $rc, $attrs );
function getLogEntriesBeforeAndList( $lc, $before, $rc, $attrs );
function getLogEntriesAfterAndList( $lc,  $after, $rc, $attrs );
function getLogEntriesBetweenAndList( $lc, $start, $end, $rc, $attrs );
function getLogEntriesBefore( $before, $rc, $attrs );
function getLogEntriesAfter( $after, $rc, $attrs );
function getLogEntriesBetween( $start, $end, $rc, $attrs );
```

The example shows how to get all existing subscription log entries.

```
$sm = $s->getSubscriptionManager();
$rc = $s->createRecipientContext();
$rmd = $rc->getMetaData();
$intAttr = $rmd->getUserAttribute( "countSendedMailings" );
$rowset = $sm->getAllLogEntries( $rc, new array($intAttr ) );
while( $rowset->next() )
{
   Print( $rowset->getDatetime() . " " . $rowset->getEmailAddress() . " "
      . $rowset->getRecipientId() . " " . $rowset->getListId() . " "
      . $rowset->getLogMessage() . " " . $rowset->getSendingId() . " " );
   if( $rowset->getRecipientState() ==
         Inx_Api_Subscription_SubscriptionLogEntryRowSet::RECIPIENT_STATE_EXISTENT )
      Print( $rowset->getInteger( $intAttr ) );
   else
      Print( "Recipient does not exists" );
}
$rowset->close();
```

3.9.2. Inx_Api_Mailing_MailingManager

The MailingManager controls all aspects concerned with mailings. To use the MailingManager for a mailing list, the `MAILING_FEATURE` has to be activated for this list.
Some of the methods exposed by `Inx_Api_Mailing_MailingManager` anticipate methods in future versions of Inxmail. Methods which have currently no function are:

```
public function requestApproval()
```

Create and Edit Mailings

```
$oMailingMgr = $oSession->getMailingManager();
$oMailing = $oMailingMgr->createMailing($oListContext);
$oMailing->updateSubject("Monthly Newsletter");
$oMailing->updateName("Monthly Newsletter");
$oMailing->commitUpdate();
```

For existing mailings, always call `lock` before updating it, and `unlock` after committing changes!
Content is put into mailings using content handlers. There are a number of such handlers:

`Inx_Api_Mailing_PlainTextContentHandler` - Handles plain text content.

`Inx_Api_Mailing_HtmlTextContentHandler` - Handles HTML-only content.

`Inx_Api_Mailing_MultiPartContentHandler` - Handles multipart content (HTML plus plain text), or mailings where their content is selected depending on the recipient profile.

`Inx_Api_Mailing_XsltMultiPartContentHandler` - Handles multipart content defined by XML/XSLT, or mailings whose content is selected depending on the recipient profile.

`Inx_Api_Mailing_XsltPlainTextContentHandler` - Handles plain text content defined by XML/XSLT.

`Inx_Api_Mailing_XsltHtmlTextContentHandler` - Handles HTML text content defined by XML/XSLT.

All of these handlers expose methods to enter the content. For example, editing a plain text mail:

```
$m->setContentHandler('Inx_Api_Mailing_PlainTextContentHandler');
$ch = $m->getContentHandler() ;
$ch->updateContent ("...any mailing content...");
```

Retrieval of Mailings

```
public function select( Inx_Api_List_ListContext $oListContext,
                        $iStateFilter );
public function select( Inx_Api_List_ListContext $oListContext,
                        $iStateFilter, $sFilter );
public function select( Inx_Api_List_ListContext $oListContext,
                        $iStateFilter, $sFilter, $iOrderAttribute,
                        $iOrderType );
```

Existing Mailings can be retrieved with the `select` methods listed above. The `Inx_Api_BOResultSets` contain `Inx_Api_Mailing_Mailing` objects. The various options define the selection and ordering criteria.

`listContext` - The mailing list to get mailings from. It is currently not possible to get mailings from multiple lists in one selection.

`stateFilter` - Select mailings by their state. `Inx_Api_Mailing_MailingManager::STATE_FILTER_ALL` matches mailings in any state. Use `Inx_Api_Mailing_Mailing::STATE_*` as single values or in bitwise combinations to select mailings by specific state(s).

`orderAttribute` - Specify the mailing attribute by which the result set is ordered. Use `Mailing.ATTRIBUTE_*`. For technical reasons, not all attributes may be used for ordering. Currently `Inx_Api_Mailing_Mailing::ATTRIBUTE_SUBJECT` and `Inx_Api_Mailing_Mailing::ATTRIBUTE_MODIFICATION_DATETIME` are possible.

`orderType` - Order direction. Use `Inx_Api_Order::ORDER_ASC` for ascending, `Inx_Api_Order::ORDER_DESC` for descending ordering.

`filter` - Free filter expression. See section below for syntax.

Filters are specified as text strings with the same syntax as Inxmail internal filters and conditions. Mailing filters are restricted to attribute - value comparisons without `AND` and `OR` combinations. Attributes are specified with the `Attribute(id)` function, where `id` corresponds to the values for the order attribute described above. A sample filter for all mailings last changed on or after Jan. 1st, 2006 is:

```
$filter = "Attribute(".
    Inx_Api_Mailing_Mailing::ATTRIBUTE_MODIFICATION_DATETIME
            .")>#01.01.2006 00:00:00#";
```

`Inx_Api_Mailing_Mailing::ATTRIBUTE_MODIFICATION_DATETIME` is a timestamp attribute, therefore a date is not sufficient, a time must also be specified. Operators and value formats are described in the Inxmail user manual, chapter 23.

Approval and Controlling Send-Out

Since Inxmail Professional API 1.6.0 it is possible to use the approval of mailings. The following methods are defined for requesting/deny/revoke approval.

```
public function approve( $iApproverId = 0, $sComment = null );
public function denyApprove( $iApproverId, $sComment );
public function requestEscalationApproval( $oEscalationDate, $oDeadline, $approverIds, $recipientIds,
        $bIsTestRecipient, $sLocale );
public function requestIdenticalApproval( $oDeadline, $approverIds, $recipientIds, $bIsTestRecipient,
        $sLocale );
public function revokeApproval( $sComment = null );
```

The methods `approve()` and `requestApproval()` are deprecated and should never used. Please use the methods above.
Following methods can be used to send mailings:

```
public function sendTestMail( $sTestAddress, $iRecipientId );

public function sendSingleMail( $iRecipientId );

public function startSending();

public function stopSending();
```

To schedule a mailing, update the schedule time. This example schedules the mailing one hour in the future:

```
$oMailing->scheduleMailing( date ('c', time()+60*60*1000 ) ) ;
```

To revoke the scheduling:

```
$oMailing->unscheduleMailing();
```

Mail Preview

Please note that starting with Inxmail Professional API version 1.11.10, the `Inx_Api_Mail_MailingRenderer` is deprecated and is replaced by the `Inx_Api_Rendering_GeneralMailingRenderer`. For more information see chapter `Inx_Api_GeneralMailing_GeneralMailingManager`.

Sending info

With the sending info you are able to get information about the sending of the mailing such as number of recipients or average mail size. For getting the sending info object call `getSendingInfo()` from the `Inx_Api_Mailing_Mailing` object.

```
interface SendingInfo
{
   public function getDeliveredMailsCount();
   public function getSentErrorCount();
   public function getBounceCount();
   public function getNotSentMailsCount();
   public function getAverageMailSize();
}
```

Starting with Inxmail Professional API version 1.11.4 you can also use the `Inx_Api_Sending_SendingHistoryManager` to access more detailled sending information. As a shortcut, you may also use the `findSendings` and `findLastSending` methods.

3.9.3. Inx_Api_TriggerMailing_TriggerMailingManager

The `Inx_Api_TriggerMailing_TriggerMailingManager` and the `Inx_Api_TriggerMailing_TriggerMailing` business object cover all aspects of the trigger mailing lifecycle. Trigger mailings were introduced with Inxmail Professional 4.2 to satisfy the need for event driven mailings. In general, trigger mailings won't be sent to all recipients of the associated list, but to a subset of the recipients, depending on the trigger conditions. The following trigger mailing types - as defined by the `Inx_Api_TriggerMailing_Descriptor_TriggerType` enumeration - are supported:

- `ACTION_MAILING`: This mailing type is triggered by an action.
- `TIME_TRIGGER_INTERVAL_MAILING`: A mailing of this type is sent to all recipients of the associated list at a freely definable interval (i.e. hourly, daily, weekly,...). The interval is described by a `Inx_Api_TriggerMailing_Descriptor_TriggerInterval` object. The interval trigger is a time trigger which is not related to a specific attribute.
- `TIME_TRIGGER_BIRTHDAY_MAILING`: A mailing of this type is sent to recipients on the annual recurrence of a specific date. A datetime attribute of the recipient acts as a baseline and the mailing is sent every year after this baseline. An offset can be specified to send the mailing

some time before or after the annual recurrence. The condition is checked once a day. The birthday trigger is an attribute driven time trigger.

- `TIME_TRIGGER_ANNIVERSARY_MAILING`: A mailing of this type is sent to recipients on the recurrence of a specific date. A datetime attribute of the recipient acts as baseline and the mailing is sent after a user defined period of time (years, months or days) after this baseline. An offset can be specified to send the mailing some time before or after the recurrence. The condition is checked once a day. The anniversary trigger is an attribute driven time trigger.
- `TIME_TRIGGER_REMINDER_MAILING`: A mailing of this type is sent to recipients on a specific date. A datetime attribute of the recipient defines that date. An offset can be specified to send the mailing some time before the date. The condition is checked once a day. The reminder trigger is an attribute driven time trigger.
- `TIME_TRIGGER_FOLLOW_UP_MAILING`: A mailing of this type is sent to recipients on a specific date. A datetime attribute of the recipient defines that date. An offset can be specified to send the mailing some time after the date. The condition is checked once a day. The follow up trigger is an attribute driven time trigger.

These basic trigger types can be used to create a wide variety of different event driven mailings. The following subsections discuss the different aspects of the trigger mailing lifecycle and how to handle them using the Inxmail Professional API.

A note for programmers who are not familiar with the concept of enumerations: Enumerations or enumerated types are basically a fixed set of named values. They are usually used to define a couple of legitimate values in a specific context and serve a purpose similar to integer constants.

The advantage of enumerations is, that you cannot specify any "weird" values because every value has to be an instance of the enumerated type. It is also possible to associate data or even behaviour (methods) with the values.

PHP does not support such a language feature like Java and C# do. In most languages the named values are a sort of constant whose value is an instance of the enumerated type. In PHP a constant cannot contain a reference type. Therefore, we implemented enumerations as classes with private constructor and methods which return the named values.

Be aware that the objects returned by the static methods are always the same object. That way it is possible to use the identity operator (===) on these objects and use them comfortably in switch statements.

Note: The `Inx_Api_TriggerMailing_TriggerMailingManager` and the `Inx_Api_Mailing_MailingManager` seem to be pretty similar (and in fact are to some degree) however, they are NOT interoperable.

Creation and editing

The heart of a trigger mailing is the `Inx_Api_TriggerMailing_Descriptor_TriggerDescriptor` which defines the trigger type and the various settings. Depending on the trigger type the mailing is either sent out by an action (action driven), on a regular basis (interval driven) or according to the value of a date attribtue (attribute driven). Interval and attribute driven triggers are also referred to as time triggers. See above for a list of the available trigger types.
It is rarely advisable to create a `Inx_Api_TriggerMailing_Descriptor_TriggerDescriptor` directly as the state space is complex and can be confusing. Generally, it's reasonable to use a `Inx_Api_TriggerMailing_Descriptor_TriggerDescriptorBuilder` for this task which will guide you through the process of creating a `Inx_Api_TriggerMailing_Descriptor_TriggerDescriptor` and complain about any missing settings and broken invariants. To obtain a builder appropriate

for the desired trigger type, use the `Inx_Api_TriggerMailing_Descriptor_TriggerDescriptorBuilderFactory`.

The following snippet exemplary shows how to create an action trigger, an interval trigger and an anniversary trigger. Be aware that in this case the most complex configuration is used. Some of the settings are optional, as documented by each builder, but this example illustrates all the capabilities of trigger mailings:

```
// obtain builder factory
$oFactory = $oSession->getTriggerMailingManager()->getTriggerDescriptorBuilderFactory();

// retrieve attributes for time triggers
$oRmd = $oSession->createRecipientContext()->getMetaData();
$iBirthdayId = $oRmd->getUserAttribute( 'Birthday' )->getId();
$counterId = $oRmd->getUserAttribute( 'Counter' )->getId();

// create end date and sending time for time triggers
$sEndDate = date('c', strtotime('+1 year'));
$sSendingTime = date('c', mktime(12, 30));

// create commands for time triggers
$oCmdFactory = $oSession->getActionManager()->getCommandFactory();
$oCmd = $oCmdFactory->createSetRelativeValueCmd( $counterId, '1' );
$aCommands = array( $oCmd );

// create action trigger
$oActionDescriptor = $oFactory->createActionTriggerDescriptorBuilder()->build();

// create interval trigger
$oIntFactory = $oSession->getTriggerMailingManager()->getTriggerIntervalBuilderFactory();
$oInterval = $oIntFactory->getWeeklyIntervalBuilder()->intervalCount( 2 )->dispatchIntervals(
    array( Inx_Api_TriggerMailing_Descriptor_TimeTriggerDispatchInterval::MONDAY(),
      Inx_Api_TriggerMailing_Descriptor_TimeTriggerDispatchInterval::FRIDAY() ) )->build();

$oIntervalDescriptor = $oFactory->createIntervalTriggerDescriptorBuilder()->startDate(
    date('c') )->sendingTime( $sSendingTime )->endDate( $sEndDate )->interval( $oInterval )
    ->attributeValueSetters( $aCommands )->build();

// create anniversary trigger
$oModificator = new Inx_Api_TriggerMailing_Descriptor_TimeTriggerOffset(
    Inx_Api_TriggerMailing_Descriptor_TimeTriggerOffsetType::WAS_AGO(),
    Inx_Api_TriggerMailing_Descriptor_TimeTriggerUnit::YEAR(), 50 );
$oOffset = new Inx_Api_TriggerMailing_Descriptor_TimeTriggerOffset(
    Inx_Api_TriggerMailing_Descriptor_TimeTriggerOffsetType::IS_IN(),
    Inx_Api_TriggerMailing_Descriptor_TimeTriggerUnit::DAY(), 1 );

$oDescriptor = $oFactory->createAnniversaryTriggerDescriptorBuilder()->startDate( date('c') )
    ->sendingTime( $sSendingTime )->endDate( $sEndDate )->attribute( $iBirthdayId )->columnModificator(
      $oModificator )->offset( $oOffset )->attributeValueSetters( $aCommands )->build();
```

The action trigger is the easiest to configure. The reason for this is simple: There is no configuration. The sending process is controlled by an action, or more specifically, an action can use and send this mailing.

The interval trigger is one of the most complex trigger types, particularly because of the need to build the interval. The trigger in the example will send the mailing every other week on Monday and Friday and will increase the Counter attribute by one. It will be active for one year from now on.

The anniversary trigger is probably the most complex attribute driven trigger type, as it offers the most settings. The trigger in the example will send the mailing to recipients who celebrate their 50th birthday the next day. It will also increase the Counter attribute by one and will be active for one year from now on.

Apart from the `Inx_Api_TriggerMailing_Descriptor_TriggerDescriptor` the creation of a trig-

ger mailing works pretty much the same way as that of a normal mailing. The following snippet shows how to create a trigger mailing that will be sent to recipients who have been a member of the associated list for one year:

```
$iOptInDate = $oSession->createRecipientContext()->getMetaData()->getSubscriptionAttribute( $oListContext )->
    getId();
$sStartDate = date('c');
$sSendingTime = date('c', mktime(12, 30));

$oTriggerMailingMgr = $oSession->getTriggerMailingManager();
$oDescriptor = $oTriggerMailingMgr->getTriggerDescriptorBuilderFactory()
    ->createAnniversaryTriggerDescriptorBuilder()->startDate( $sStartDate )->sendingTime( $sSendingTime )
    ->attribute( $iOptInDate )->columnModificator( new Inx_Api_TriggerMailing_Descriptor_TimeTriggerOffset(
        Inx_Api_TriggerMailing_Descriptor_TimeTriggerOffsetType::WAS_AGO(),
        Inx_Api_TriggerMailing_Descriptor_TimeTriggerUnit::YEAR(), 1 ) )->build();

$oMailing = $oTriggerMailingMgr->createTriggerMailing( $oListContext, $oDescriptor );
$oMailing->updateName( 'One year anniversary' );
$oMailing->updateSubject( "Thank's for staying with us!" );
$oMailing->commitUpdate();
```

As mentioned before, action mailings work slightly different. Instead of configuring the sending process inside the `Inx_Api_TriggerMailing_Descriptor_TriggerDescriptor` it is entirely controlled by an action. In order to use an action mailing you will have to add a `Inx_Api_Action_SendActionMailCommand` to an action. The following snippet shows how to create an action mailing and an action which sends the mailing:

```
// create action mailing
$oTmm = $oSession->getTriggerMailingManager();
$oDescriptor = $oTmm->getTriggerDescriptorBuilderFactory()
    ->createActionTriggerDescriptorBuilder()->build();
$oMailing = $oTmm->createTriggerMailing( $oListContext, $oDescriptor );
$oMailing->updateName( 'Snippet Action Mailing' );
$oMailing->updateSubject( 'Snippet Action Mailing' );
$oMailing->commitUpdate();

// action mailing must be approved
$oMailing->approveImmediately( 'The mailing is approved.' );

// create action
$oAm = $oSession->getActionManager();
$oAction = $oAm->createAction( $oListContext );
$oAction->updateEventType( Inx_Api_Action_Action::EVENT_TYPE_SUBSCRIBE );
$oAction->updateName( 'Snippet Action' );

// create command
$oCf = $oAm->getCommandFactory();
$aCmds = array( $oCf->createSendActionMailCmd( $oListContext->getId(), $oMailing->getId() ) );
$oAction->updateCommands( $aCmds );
$oAction->commitUpdate();
```

For existing trigger mailings, always call `lock` before updating it, and `unlock` after committing changes! Content is put into trigger mailings using content handlers. There are a number of such handlers:

`Inx_Api_Mailing_PlainTextContentHandler` - Handles plain text content.

`Inx_Api_Mailing_HtmlTextContentHandler` - Handles HTML-only content.

`Inx_Api_Mailing_MultiPartContentHandler` - Handles multipart content (HTML plus plain text), or mailings whose content is selected depending on the recipient profile.

`Inx_Api_Mailing_XsltMultiPartContentHandler` - Handles multipart content defined by XML/XSLT, or mailings whose content is selected depending on the recipient profile.

`Inx_Api_Mailing_XsltPlainTextContentHandler` - Handles plain text content defined by XML/XSLT.

`Inx_Api_Mailing_XsltHtmlTextContentHandler` - Handles HTML text content defined by XML/XSLT.

All of these handlers offer methods to update content. The following snippet exemplary shows how to edit a plain text trigger mail:

```
$m->setContentHandler('Inx_Api_Mailing_PlainTextContentHandler');
$ch = $m->getContentHandler() ;
$ch->updateContent ("...any mailing content...");
```

Retrieval

The `Inx_Api_TriggerMailing_TriggerMailingManager` offers several methods to retrieve trigger mailings. Most of them are used to retrieve a set of trigger mailings matching a specific condition. Given the ID of the mailing is known, the `get` method can be used to retrieve a single trigger mailing. Here is a list of the available methods:

```
public function get( $iId );
public function selectAll();
public function selectByState( Inx_Api_List_ListContext $listContext, Inx_Api_TriggerMailing_StateFilter $stateFilter,
  Inx_Api_TriggerMailing_TriggerMailingAttribute $orderAttribute = null, $iOrderType = null, $sFilter = null );
```

Existing trigger mailings can be retrieved with the `select` methods listed above. The `Inx_Api_BOResultSets` contain `Inx_Api_TriggerMailing_TriggerMailing` objects. Options define the selection and ordering criteria:

`listContext` - The mailing list to retrieve trigger mailings from. It is currently not possible to retrieve trigger mailings from multiple lists in one selection.

`stateFilter` - Selects trigger mailings by their mailing and/or trigger state.

`orderAttribute` - Specifies the trigger mailing attribute by which the result set is ordered. Use `Inx_Api_TriggerMailing_TriggerMailingAttribute::*()`. For technical reasons, not all attributes may be used for ordering. Currently, the following attributes may be used:

- `SUBJECT`
- `NAME`
- `SINGLE_SEND_COUNT`
- `ACTIVATION_DATETIME`
- `MODIFICATION_DATETIME`

`orderType` - Order direction. Use `Inx_Api_Order::ORDER_ASC` for ascending and `Inx_Api_Order::ORDER_DESC` for descending ordering.

`filter` - Free filter expression.

Using a `Inx_Api_TriggerMailing_StateFilter` trigger mailings can be retrieved according to their state. A trigger mailing has two types of states: the mailing state and the trigger state. The mailing state reflects the state of the mailing, pretty much like the state of a normal mailing. The possible values are defined by the `Inx_Api_TriggerMailing_TriggerMailingState` enumeration. The trigger state, on the other hand, reflects the state of the trigger which can be active or inactive. The possible values are defined by the `Inx_Api_TriggerMailing_TrigerState` enumeration.

A `Inx_Api_TriggerMailing_StateFilter` consists of a combination of both filter types. A trigger mailing must match at least one of the specified mailing types and the trigger type. However, it is possible to create state filters that match any mailing and/or trigger state. A state filter that matches any mailing and trigger state is referred to as 'all matching state filter' which

can be obtained from the manager as Singleton. The following methods can be used to create a `Inx_Api_TriggerMailing_StateFilter`:

```
public function createMailingStateFilter( array $stateFilter = null );
public function createTriggerStateFilter( Inx_Api_TriggerMailing_TriggerState $stateFilter = null );
public function createStateFilter( array $mailingStateFilter = null, Inx_Api_TriggerMailing_TriggerState $triggerStateFilter =
    null );
public function createAllMatchingStateFilter();
```

Using the appropriate method, it is easy to create a `Inx_Api_TriggerMailing_StateFilter` which matches a set of mailing states and/or trigger state or to retrieve all trigger mailings of a list, disregarding their state using the all matching state filter. For an example of how to create a `Inx_Api_TriggerMailing_StateFilter`, see the retrieval snippet at the end of this section.

Free filter expressions are specified as text strings with the same syntax as Inxmail internal filters and conditions. Trigger mailing filters are restricted to attribute - value comparisons without `AND` and `OR` combinations (only a single attribute may be matched). Attributes are specified with the `Attribute(id)` function, where `id` corresponds to the id of any attribute defined in the `Inx_Api_TriggerMailing_TriggerMailingAttribute` enumeration. An examplary filter expression is shown in the retrieval snippet at the end of this section. The available operators and value formats of filter expressions are described in the Inxmail user manual, chapter 23.

The following snippet shows how to retrieve all trigger mailings of the specified list which are in the `DRAFT` or `APPROVAL_REQUESTED` state and have been edited during the last hour. The snippet prints out the mailing name in ascending alphabetical order.

```
$oTriggerMailingMgr = $oSession->getTriggerMailingManager();

$sFilterDate = date( 'd.m.Y H:i:s', strtotime( '-1 hour' ) );
$sFilter = 'Attribute(' . Inx_Api_TriggerMailing_TriggerMailingAttribute::MODIFICATION_DATETIME()->getId() . ') > #'
    . $sFilterDate . '#';

$aMailingStateFilter = array( Inx_Api_TriggerMailing_TriggerMailingState::DRAFT(),
    Inx_Api_TriggerMailing_TriggerMailingState::APPROVAL_REQUESTED() );
$oStateFilter = $oTriggerMailingMgr->createMailingStateFilter( $aMailingStateFilter );

$oSet = $oTriggerMailingMgr->selectByState( $oListContext, $oStateFilter,
    Inx_Api_TriggerMailing_TriggerMailingAttribute::NAME(), Inx_Api_Order::ASC, $sFilter );

for( $i = 0; $i < $oSet->size(); $i++ )
{
  $oMailing = $oSet->get( $i );
  echo $oMailing->getName() . '<br>';
}

$oSet->close();
```

Approval and controlling send-out

The approval process of trigger mailings is almost identical to that of regular mailings with two exceptions: the deprecated methods were removed and a new method for the immediate approval of trigger mailings was added. The following methods are available to manage the approval process:

```
public function approveImmediately( $sComment );
public function approve( $iApproverId, $sComment );
public function denyApprove( $iApproverId, $sComment );
public function requestEscalationApproval( $sEscalationDate, $sDeadline, array $approverIds, array $recipientIds,
  $blIsTestRecipient, $sLocale );
public function requestIdenticalApproval( $sDeadline, array $approverIds, array $recipientIds,
  $blIsTestRecipient, $sLocale );
public function revokeApproval( $sComment = null );
```

The normal approval workflow requires an approval request in which the user decides whether the approval is granted or denied. There are two different types of approval requests: escalating and identical.

The escalating approval process involves only the primary approver at first. Only if the primary approver does not respond to the request by a given escalation date, the secondary approver will get involved. The identical approval process involves both approvers immediately and requires both to grant the approval.

Revoking the approval is possible during the request or after the approval. It is also possible to bypass the normal approval process by approving the trigger mailing immediately. Be aware that this requires the corresponding right.

The following snippet shows how to implement the normal approval workflow:

```
$sEscalationDate = date( 'c', strtotime( '+7 days' ) );
$sDeadline = date( 'c', strtotime( '+14 days' ) );

$aApproverIds = array( $iPrimaryId, $iSecondaryId );
$aRecipientIds = array( $iRecipientId );

$oMailing->requestEscalationApproval( $sEscalationDate, $sDeadline, $aApproverIds, $aRecipientIds, false, 'en' );
$oMailing->approve( $iPrimaryId, 'Looks good!' );
```

Sending trigger mailings differs gravely from sending normal mailings. While normal mailings are sent only once to every recipient of the associated list, trigger mailings are sent to a subset of these recipients on a regular basis, depending on the trigger. This is the reason why it is not possible (and makes no sense) to schedule trigger mailings or start sending them manually. Instead, a trigger mailing is *activated* or *deactivated* using the following methods:

```
public function activateSending();
public function deactivateSending( $blStopActiveSending );
```

Mail preview

Please note that starting with Inxmail Professional API version 1.11.10, the `Inx_Api_TriggerMail_TriggerMailingRenderer` is deprecated and is replaced by the `Inx_Api_Rendering_GeneralMailingRenderer`. For more information see chapter `Inx_Api_GeneralMailing_GeneralMailingManager`.

Sending info

To retrieve the date of the next sending interval, use the `getNextSending()` method.

Starting with Inxmail Professional API version 1.11.4 you can also use the `Inx_Api_Sending_SendingHistoryManager` to access more detailled sending information. As a shortcut, you may also use the `findSendings` and `findLastSending` methods.

3.9.4. Inx_Api_GeneralMailing_GeneralMailingManager

Introduced in the Inxmail Professional API version 1.11.10, the `Inx_Api_GeneralMailing_GeneralMailingManager` provides read-only access to most of the mailing types supported by Inxmail Professional. In contrast to the other mailing managers, the `Inx_Api_GeneralMailing_GeneralMailingManager` employs a single interface.

The following mailing types are currently supported by the `Inx_Api_GeneralMailing_GeneralMailingManager`:

- Regular mailings

- Action mailings
- Time trigger mailings (like birthday mailing and interval mailing)
- Subscription trigger mailings
- Split test mailings
- Sequence mailings

This is helpful especially if you want to aggregate data from various mailings of different types. Without the `Inx_Api_GeneralMailing_GeneralMailingManager` you would have to use several mailing managers and aggregate the data they produce. Also, the `Inx_Api_GeneralMailing_GeneralMailingManager` for the first time offers access to split test mailings, sequence mailings and subscription trigger mailings.

Retrieval of GeneralMailings

The `Inx_Api_GeneralMailing_GeneralMailingManager` offers the following retrieval methods:

```
public function get( $iId );
public function selectAll();
public function createQuery();
```

Aside from the usual retrieval methods provided by all `Inx_Api_ROBOManager`s, there is a wide range of criteria which can be freely combined using the `Inx_Api_GeneralMailing_GeneralMailingQuery` to find `Inx_Api_GeneralMailing_GeneralMailing`s.

The `Inx_Api_GeneralMailing_GeneralMailingQuery` implements a fluent interface for creating and executing queries. The basic idea is to simply create a query object and combine the available filters as you need them instead of figuring out which method offers the appropriate set of filters. This allows you to create complex queries, while the fluent interface keeps the syntax as concise as possible, thus producing more readable and maintainable code.

The following criteria are supported by `Inx_Api_GeneralMailing_GeneralMailingQuery`:

- The mailing type
- The ID of the list containing the mailing
- The mailing ID
- The mailing name
- The mailing subject
- The creation date of the mailing
- The last modification date of the mailing

Each of these criteria can be specified as an array of multiple values. A mailing matches the query if:

1. All criteria are met (AND concatenated)
2. For each of the criteria at least one value matches (OR concatenated)

Furthermore, it is possible to sort the output of the query in either ascending or descending order by one of the following attributes:

- The mailing ID

- The mailing type
- The ID of the list containing the mailing
- The mailing name
- The mailing subject
- The creation date of the mailing
- The last modification date of the mailing

The following snippet demonstrates a very simple, yet quite effective query which retrieves all mailings with the specified IDs:

```
$oGeneralMailingManager = $oSession->getGeneralMailingManager();
$oGeneralMailingQuery = $oGeneralMailingManager->createQuery();

$aIds = array( 1, 2, 3 );
$oResult = $oGeneralMailingQuery->mailingIds( $aIds )->executeQuery();

foreach( $oResult as $oGeneralMailing )
{
  echo $oGeneralMailing->getName() . '<br>';
}

if(null != $oResult)
{
  $oResult->close();
}
```

Of course you can also create much more complex queries, like the one presented in the following snippet:

```
$oGeneralMailingManager = $oSession->getGeneralMailingManager();
$oGeneralMailingQuery = $oGeneralMailingManager->createQuery();

$aMailingTypes = array(Inx_Api_GeneralMailing_MailingType::REGULAR_MAILING(),
  Inx_Api_GeneralMailing_MailingType::TIME_TRIGGER_MAILING());
$aListIds = array( 3, 5, 7 );
$aNames = array( 'Spring Campaign', 'Autumn Campaign' );
$aSubjects = array( 'Good news', 'Bad news' );

$oResult = $oGeneralMailingQuery->mailingTypes( $aMailingTypes )->listIds( $aListIds )->names( $aNames )
  ->subjects( $aSubjects )->sort( Inx_Api_GeneralMailing_GeneralMailingAttribute::LIST_ID(),
  Inx_Api_Order::ASC )->executeQuery();

foreach( $oResult as $oGeneralMailing )
{
  $listId = $oGeneralMailing->getListContextId();
  $name = $oGeneralMailing->getName();
  $subject = $oGeneralMailing->getSubject();

  echo "$listId: $name / $subject<br>";
}

if(null != $oResult)
{
  $oResult->close();
}
```

This query retrieves all mailings which:

1. Are either regular mailings **or** time trigger mailings **and**
2. Reside in list 3 **or** 5 **or** 7 **and**
3. Whose name is either "Spring Campaign" **or** "Autumn Campaign" **and**

4. Whose subject is either "Good news" **or** "Bad news"

The result is ordered by the ID of the lists containing the mailings in ascending order.

The GeneralMailing BusinessObject

The `Inx_Api_GeneralMailing_GeneralMailing` business object provides some basic data for a mailing:

- The mailing ID
- The mailing name
- The mailing subject
- The ID of the list containing the mailing
- The mailing type
- The creation date of the mailing
- The last modification date of the mailing
- All sendings of the mailing
- The last sending of the mailing

Rendering & Preview

To render a mailing or create a preview of it, use the `Inx_Api_Rendering_GeneralMailingRenderer`. As of Inxmail Professional API version 1.11.10, the `Inx_Api_Rendering_GeneralMailingRenderer` replaces the renderers formerly used for mailings and trigger mailings. It can be used to render mailings of the following types:

- Regular mailings
- Action mailings
- Time trigger mailings
- Subscription trigger mailings
- Split test mailings
- Sequence mailings

Terminology note: In the context of this guide, the term rendering refers to the process of producing the actual HTML and plain text parts of a mailing. This process consists of the following steps:

1. Parsing the Inxmail Professional specific mailing code
2. Performing certain transformations
3. Personalizing the content for a specific recipient
4. Producing the HTML and plain text parts as they would be present in a sent mailing

To render a mailing, you need to acquire an instance of `Inx_Api_Rendering_GeneralMailingRenderer` from the `Inx_Api_GeneralMailing_GeneralMailingManager`. The rendering is a two-stage process. First, you need to parse a mailing in a specific build mode. Afterwards, you need to build it for a specific recipient. The following snippet demonstrates this process:

```
$oRenderer = $oSession->getGeneralMailingManager()->createRenderer();
$oRenderer->parse($iMailingId, Inx_Api_Rendering_BuildMode::ALTERNATIVEVIEW_ACTIVE());
$oContent = $oRenderer->build($iRecipientId);

if ($oRenderer != null)
  $oRenderer->close();
```

As briefly mentioned above, you need to specify a build mode during the parse stage of the rendering process. The available build modes are specified in the `Inx_Api_Rendering_BuildMode` enumeration:

- `NORMAL` - Mode for generating a normal mailing, ready to be sent.
- `ALTERNATIVEVIEW_ACTIVE` - Mode for alternative view. All links are fully functional. Embedded images are replaced with http references to image resources on the Inxmail server.
- `ALTERNATIVEVIEW_INACTIVE` - Mode for alternative view. Standard links are fully functional, tracking links are functional but will not trigger any event or generate any click. Embedded images are replaced with http references to image resources on the Inxmail server.
- `PREVIEW` - Mode for mail preview. Standard links are fully functional, tracking links are functional but will not trigger any event or generate any click, unsubscription links will redirect but not unsubscribe anybody. Embedded images are replaced with http references to image resources on the Inxmail server. The function `InInboxView()` will return true while building the mailing.
- `ARCHIVE` - Mode for archive view. Standard links are fully functional, tracking links are functional but will not trigger any event or generate any click, unsubscription links will redirect but not unsubscribe anybody. Embedded images are replaced with http references to image resources on the Inxmail server. The function `InInboxView()` will return true while building the mailing.
- `ALTERNATIVEVIEW_ACTIVE_SIMPLE_LINKS` - Mode for alternative view. All links are fully functional but converted to simple links. Embedded images are replaced with http references to image resources on the Inxmail server.
- `NEWSLETTER_SIMPLE_LINKS` - All links are fully functional but converted to simple links. Embedded images are replaced with http references to image resources on the Inxmail server. The function `InInboxView()` will return true while building the mailing.

The `build` method returns an instance of `Inx_Api_Rendering_Content` which contains all relevant data of the rendered mailing:

- The content type (which is the MIME type)
- The rendered, personalized HTML text part, if any
- The rendered, personalized plain text part, if any
- The personalized subject
- The email address of the recipient
- The email address of the sender
- The reply-to address
- The bounce address
- The, possibly personalized, attachments
- The embedded images
- The header information

Attachments and embedded images are conveyed in an instance of class `Inx_Api_Rendering_Attachment`. This object offers the following information:

- The file name or embedded image identifier
- The content type (which is the MIME type)
- The size in bytes
- An input stream which can be used to download the file

The following snippet demonstrates how to extract some key data of the content:

```
// Now the content can be accessed:
echo "From: " . $oContent->getSenderAddress() . "<br>";
echo "To:" . $oContent->getRecipientAddress() . "<br>";
echo "Reply-To:" . $oContent->getReplyToAddress() . "<br>";
echo "Additional Headers: ";
print_r($oContent->getHeader()) . "<br>";
echo "Content:<br>" . $oContent->getPlainText();
```

3.9.5. Inx_Api_SplitTest_SplitTestManager and Inx_Api_SplitTestMailing_SplitTestMailingManager

Introduced in the Inxmail Professional API version 1.13.1, the `Inx_Api_SplitTest_SplitTestManager` and `Inx_Api_SplitTestMailing_SplitTestMailingManager` provide read-only access to `Inx_Api_SplitTest_SplitTest` and `Inx_Api_SplitTestMailing_SplitTestMailing` objects. This is helpful especially if you want to aggregate all split test mailings that refer to the same split test.

Retrieval of SplitTests and SplitTestMailings

The `Inx_Api_SplitTest_SplitTestManager` offers the usual retrieval methods provided by all `Inx_Api_BOManagers`:

```
public function get( $iId );
public function selectAll();
```

The same is true for the `Inx_Api_SplitTestMailing_SplitTestMailingManager`:

```
public function get( $iId );
public function selectAll();
```

It is important to note that although `Inx_Api_SplitTest_SplitTestManager` and `Inx_Api_SplitTestMailing_SplitTestMailingManager` inherit from the `Inx_Api_BOManager` class, all write access methods (`remove`, `commitUpdate`) are currently not supported and throw a '**Not Implemented**' exception.

The `Inx_Api_SplitTest_SplitTest` business object provides the following data:

- The split test ID
- The split test name

The `Inx_Api_SplitTestMailing_SplitTestMailing` business object provides nearly the same data for a split test mailing as the according GeneralMailing Objects, with the exception of an additional SplitTest attribute:

- The mailing ID
- The mailing name
- The mailing subject

- The ID of the list containing the mailing
- **The SplitTest the mailing belongs to**
- The creation date of the mailing
- The last modification date of the mailing
- All sendings of the mailing
- The last sending of the mailing

While most of these methods return immediately, be aware that the `getSplitTest` method performs an additional server call.

3.9.6. Inx_Api_DesignTemplate_DesignCollectionManager

With this `Inx_Api_DesignTemplate_DesignCollectionMananger` there is a direct Api access to `Inx_Api_DesignTemplate_DesignCollection`s. You can import them and get access to the informations which collections are available on the system. You can import itc files in a certain `Inx_Api_Recipient_ListContext` and get accesss to the readonly interface of the `Inx_Api_DesignTemplate_DesignCollection`s.
This is achieved via a ResultSet which contains the desired `Inx_Api_DesignTemplate_DesignCollection`s. With the Informations gained by this methods you can generate new `Inx_Api_Mailing_Mailing`s via the `Inx_Api_Mailing_MailingManager`.

Note: This is a readonly access!

This sample shows how to generate a mailing with a newly imported design collection:

```
$oMailing = $oSession->getMailingManager()->createMailing($lc) ;
$oMailing->setContentHandler( 'Inx_Api_Mailing_XsltMultiPartContentHandler' );

$oDesignCollectionManager = $oSession->getDesignCollectionManager();
$oListContext = $oSession->getListContextManager()->findByName("Name of List");

$stream = file('test.itc', 'rb');
$oCollection = $oDesignCollectionManager->importDesignCollection($stream ,$oListContext);
fclose($stream);

$aTemplates = $oCollection->getTemplates();
$oXsltMultiPartContentHandler = $oMailing->getContentHandler();
$aStyles = $aTemplates[0]->getHTMLStyles();
$oXsltMultiPartContentHandler->updateStyle($aStyles[0]);
$oMailing->commitUpdate() ;
```

This sample shows how to list all available styles in all `Inx_Api_DesignTemplate_DesignCollection` in a certain `Inx_Api_List_ListContext`.

```
$oDesignCollectionManager = $oSession->getDesignCollectionManager();
$oListContext = $oSession->getListContextManager()->findByName("Name of List");
$oSet = $oDesignCollectionManager->select($oListContext) ;
for ( $i=0; $i<$oSet->size(); $i++)
{
    $oDesignCollection = $oSet->get($i);
    echo $oDesignCollection->getVendor()."\n";
    echo $oDesignCollection->getVendorURL() . "\n";

    $aTemplates = $oDesignCollection->getTemplates();
    for ($j = 0; $j < count($aTemplates); $j++)
    {
        $oTemplate = $aTemplates[$j] ;
        echo $oTemplate->getName()."\n";
        echo $oTemplate->getId()."\n";
        $aHtmlStyles = $oTemplate->getHTMLStyles();
        for($k=0; $k < count($aHtmlStyles); $k++)
        {
            echo $aHtmlStyles[$k]->getTemplateID()."\n";
            echo $aHtmlStyles[$k]->getStyleName()."\n";
        }
    }
}
```

3.9.7. Inx_Api_MailingTemplate_MailingTemplateManager

With this `Inx_Api_MailingTemplate_MailingTemplateManager` there is a direct Api access to `Inx_Api_MailingTemplate_MailingTemplate`s. You can create them and retrieve them via this Manager.
This sample shows how to generate a new `Inx_Api_MailingTemplate_MailingTemplate` and updates the name of it.

```
$oMailingTemplateManager = $oSession->getMailingTemplateManager();
$oListContext = $oSession->getListContextManager()->findByName ("Name of List");
$oMailingTemplate = $oMailingTemplateManager->createTemplate ( $oListContext ,
            Inx_Api_MailingTemplate_MailingTemplate::MIME_TYPE_HTML_TEXT );

$oMailingTemplate->updateName("Desired name");
$oMailingTemplate->commitUpdate();
```

3.9.8. Inx_Api_TextModule_TextmoduleManager

With this `Inx_Api_TextModule_TextmoduleManager` there is a direct Api access to `Inx_Api_TextModule_Textmodule`s. You can create them and retrieve them via this Manager.
This sample shows how to generate a new `Inx_Api_TextModule_Textmodule` and updates the name of it.

```
$oTextmoduleManager = $oSession->getTextmoduleManager();
$oListContext = $oSession->getListContextManager()->findByName( "Name of List" );
$oTextmodule = $oTextmoduleManager->createTextmodule( $oListContext,
              Inx_Api_TextModule_TextModule::MIME_TYPE_HTML_TEXT );

$oTextmodule->updateName("Desired name");
$oTextmodule->commitUpdate();
```

3.9.9. Inx_Api_Transformation_TransformationManager

The `Inx_Api_Transformation_TransformationManager` provides access to the data source transformations used by the Inxmail Professional content agent.

A transformation is used to transform the data provided by a data source into HTML content that can be embedded in a mailing. To achieve this, the transformation applies a previously defined XSL

transformation on the XML data provided by the data source. To embed the transformed content in a mailing, use the content-include tag and provide the name of the data source as well as the name of the transformation to be applied on the content.

The `Inx_Api_Transformation_TransformationManager` can be used to retrieve a single transformation by id or to retrieve all registered transformations. You can also create your own transformation or edit an existing one.

Retrieval of transformations

The `Inx_Api_Transformation_TransformationManager` offers the usual retrieval methods provided by all `Inx_Api_BOManagers`:

```
public function get( $iId );
public function selectAll();
```

Creating transformations

To create a `Inx_Api_Transformation_Transformation`, you need to provide a name and the actual XSL transformation. The following snippet demonstrates how to create a transformation:

```
$sSampleXsl = '<pseudo xslt><transform><something>text text</something></transform></pseudo xslt>';

$oTransformationManager = $oSession->getTransformationManager();
$oTransformation = $oTransformationManager->createTransformation( 'Name Of XSLT Tranformation' );
$oTransformation->updateXslt( $sSampleXsl )->commitUpdate();
```

Please note that for brevity this example does not use a valid XSL transformation. For more information on XSLT, see the W3C recommendation. Also, be aware that the name has to be unique. Attempting to create a transformation with the same name as an existing one will trigger an `Inx_Api_UpdateException`.

Editing transformations

The following snippet demonstrates how to assign a different XSLT to a `Inx_Api_Transformation_Transformation`:

```
$sUpdateXsl = '<changed xslt><transform><something>text text</something></transform></changed xslt>';

$oTransformationManager = $oSession->getTransformationManager();
$oTransformation = $oTransformationManager->get( $iTransformationId );
$oTransformation->updateXslt( $sUpdateXsl )->commitUpdate();
```

Please note that it is not possible to modify the name of a transformation after it was created. This is due to the fact that transformations are referenced by name inside of mailings. Modifying the name of a transformation that is already in use would break existing mailings.

3.9.10. Inx_Api_DataAccess_DataAccess

With this `Inx_Api_DataAccess_DataAccess` there is a direct Api access to read link or click data. There are two types of objects to get the preferred data. One is the `Inx_Api_DataAccess_LinkData` object. With this object there can be searched for link data by recipient id, mailing id or link id. The other object is the `Inx_Api_DataAccess_ClickData`. Which is used for searching click data by recipient id, mailing id, both or link id. Both objects returning a row set. With this row set it can be easily navigated through the result set.

LinkData

It is important to note that a link can be permanent or temporary. Temporary links are created each time you create a preview of a mailing, either using the Inxmail Professional API, the Inxmail Professional Client application or one of the mailing related JSPs (e.g. HTML mail or archive) shipped

with the software. These links do not trigger any events and are removed once the mailing is sent.

Permanent links on the other hand are created for each sending of a mailing. They do trigger events and will not be deleted as long as the mailing that contains them exists. This implies that permanent links actually are removed once the mailing that contains them is deleted.

You can decide whether you wish to retrieve all links (permanent and temporary) or if you prefer to retrieve permanent links only. The following methods always retrieve all links:

- `selectByMailing(int)`
- `selectByLink(int)`
- `selectByRecipient(int)`
- `selectByLinkName(String)`

The following methods retrieve all links or permanent links only, depending on the `permanentLinksOnly` boolean parameter:

- `selectByMailing(int, boolean)`
- `selectByLinkName(String, boolean)`

Please note, that there is no such method for retrieval by link and recipient. Retrieval by link makes the parameter useless, as you already specify the specific link you are interested in. Retrieval by recipient always returns permanent links only because temporary links do not generate any clicks which would be necessary to establish the connection between link and recipient.

Below is a sample for getting all link data for a given recipient id.

```
$oDataAccess = $oSession->getDataAccess();
$oLinkData = $oDataAccess->getLinkData();
...
$oLinkDataRowSet = $oLinkData->selectByRecipient( $id );
```

Fluent interface for links

In Inxmail Professional API 1.12.1, a new fluent interface for retrieving link data was introduced. The basic idea is to simply create a query object and combine the available filters as you need instead of figuring out which method offers the appropriate set of filters. This allows you to create complex queries, while the fluent interface keeps the syntax as concise as possible, thus producing more readable and maintainable code.

Using the new fluent query interface, you can filter the link data by link ID, link name, link type, mailing ID and recipient ID. By default, a query will set a filter for permanent links only. It is possible to override this filter in order to retrieve temporary links as well.

Be aware though, that you have to construct your queries careful with respect to the amount of links fetched by the query. For more information on this topic and the limitations of the query interface, see section *Performance considerations*.

The following sample demonstrates one of the simplest and most common link data queries: retrieving all temporary and permanent links of type unique count having the link name "New product" or "Old product".

```
$aLinkTypes = array( Inx_Api_DataAccess_LinkDataRowSet::LINK_TYPE_UNIQUE_COUNT );
$aLinkNames = array( 'New product', 'Old product' );

$oLinkDataQuery = $oSession->getDataAccess()->getLinkDataWithNewLinkType()->createQuery();
$oLinkDataRowSet = $oLinkDataQuery->permanentAndTemporaryLinks()->linkTypes( $aLinkTypes )->linkNames(
    $aLinkNames )->executeQuery();
```

Performance considerations
When using the new `Inx_Api_DataAccess_LinkDataQuery`, you need to be aware of the fact that all these new filter possibilities and combinations come at a price: you need to be careful to make your filter conditions as narrow as possible.

With the new fluent style API it is very easy to retrieve all links of the system at once. This is not advisable, though, due to the sheer amount of links that could be present in the target system. This large number of links produces two problems:

1. The ID of each and every link needs to be read from the database during the initial fetch which in this case is a lot of data.
2. Because there are so many links involved, iterating over the `Inx_Api_DataAccess_LinkDataRowSet` will naturally take quite some time.

Huge numbers of links can cause memory problems
Issue number one is the more critical one because this huge amount of IDs needs to be stored in-memory to support the necessary pagination of the `Inx_Api_DataAccess_LinkDataRowSet`. If you have, say, one billion links in your system and each ID takes up four bytes of memory, this would make a total of four billion bytes which is roughly 3.8 gigabytes for the IDs only.

The number of links retrievable in one call is limited
You have a safety net though: the Inxmail Professional server will terminate any `Inx_Api_DataAccess_LinkDataQuery` request that produces an overall result size of over ten million links, by default. Any request with a result size above this threshold will result in a server-side `RuntimeException`.

Use a smart synchronization strategy
On the other hand there are very rare occasions where you would actually need to fetch all of the links at once. Most of the time you will probably be interested in all links associated to a mailing or list. If you are intending to synchronize all links we strongly encourage you to use a pagination mechanism which is only fetching the links which were changed since the last synchronization. In order to do so, you will have to determine the changed mailings in the first place. Be careful to keep the number of links fetched per request below a reasonable limit by applying appropriate filter conditions.

Close your row sets
One final word regarding `Inx_Api_DataAccess_LinkDataRowSet`: Be sure to close these resources once you have read all of the links and try to avoid keeping multiple `Inx_Api_DataAccess_LinkDataRowSets` alive simultaneously. The ID list on the server is stored until you close either the row set or the session. If you do neither of these, it will be discarded once the session is marked as inactive. Do *not* rely on this fact because the data will accumulate pretty fast depending on the amount of data you are synchronizing.

ClickData

This sample shows how to get all click data for a given recipient id.

```
$oDataAccess = $oSession->getDataAccess();
$oClickData = $oDataAccess->getClickData();
$oRecipientContext = $oSession->createRecipientContext();
$oEmail = $oRecipientContext->getMetaData()->getEmailAttribute();
...
$oClickDataRowSet = $oClickData->selectByRecipient( $id ,
            $oRecipientContext , array($oEmail));
```

Fluent interface for clicks

In Inxmail Professional API 1.11.4, a new fluent interface for retrieving click data was introduced. The basic idea is to simply create a query object and combine the available filters as you need instead of figuring out which method offers the appropriate set of filters. This allows you to create complex queries, while the fluent interface keeps the syntax as concise as possible, thus producing more readable and maintainable code.

Using the new fluent query interface, you can now filter the click data by link type, which for example enables you to search for all clicks on unique count links. You can also retrieve all clicks filtered only by date. Furthermore, it is now possible to filter by more than one mailing ID, link ID, recipient ID and sending ID, thus giving you greater freedom to create even more complex queries.

Be aware though, that you have to construct your queries careful with respect to the amount of clicks fetched by the query. For more information on this topic and the limitations of the query interface, see section *Performance considerations*.

The following sample demonstrates one of the simplest and most common click data queries: retrieving all clicks which have been performed since yesterday. Note that the last two lines show the actual query.

```
$oRecipientContext = $oSession->createRecipientContext();
$aAttrs = array( $oRecipientContext->getMetaData()->getEmailAttribute() );
$sStart = date( 'c', strtotime( '-1 day' ) );

$oClickDataQuery = $oSession->getDataAccess()->getClickData()->createQuery( $oRecipientContext, $aAttrs );
$oClickDataRowSet = $oClickDataQuery->after( $sStart )->executeQuery();
```

To demonstrate the power and conciseness of the fluent query interface, the following sample shows how to retrieve all clicks for a set of mailings, recipients and link types which were performed during February 2013.

```
$oRecipientContext = $oSession->createRecipientContext();
$aAttrs = array( $oRecipientContext->getMetaData()->getEmailAttribute() );
$sStart = date( 'c', mktime( 0, 0, 0, 2, 1, 2013 ) );
$sEnd = date( 'c', mktime( 23, 59, 59, 2, 23, 2013 ) );

$aMailingIds = array( 1234, 4711 );
$aRecipientIds = array( 2, 3, 5, 7, 11, 13, 17 );
$aLinkTypes = array( Inx_Api_DataAccess_LinkDataRowSet::LINK_TYPE_UNIQUE_COUNT,
    Inx_Api_DataAccess_LinkDataRowSet::LINK_TYPE_OPENING_COUNT );

$oClickDataQuery = $oSession->getDataAccess()->getClickData()->createQuery( $oRecipientContext, $aAttrs );
$oClickDataRowSet = $oClickDataQuery->mailings( $aMailingIds )->recipients( $aRecipientIds )->linkTypes(
    $aLinkTypes )->between( $sStart, $sEnd )->executeQuery();
```

Performance considerations

When using the new `Inx_Api_DataAccess_ClickDataQuery`, you need to be aware of the fact that all these new filter possibilities and combinations come at a price: you need to be careful to make your filter conditions as narrow as possible.

With the new fluent style API it is very easy to retrieve all clicks of the system at once. This is

not advisable, though, due to the sheer amount of clicks that could be present in the target system. This large number of clicks produces two problems:

1. The ID of each and every click needs to be read from the database during the initial fetch which in this case is a lot of data.
2. Because there are so many clicks involved, iterating over the `Inx_Api_DataAccess_ClickDataRowSet` will naturally take quite some time.

Huge numbers of clicks can cause memory problems
Issue number one is the more critical one because this huge amount of IDs needs to be stored in-memory to support the necessary pagination of the `Inx_Api_DataAccess_ClickDataRowSet`. If you have, say, a billion clicks in your system and each ID takes up four bytes of memory, this would make a total of four billion bytes which is roughly 3.8 gigabytes! Needless to say this is too much to keep in memory.

The number of clicks retrievable in one call is limited
You have a safety net though: the Inxmail Professional server will terminate any `Inx_Api_DataAccess_ClickDataQuery` request that produces an overall result size of over ten million clicks, by default. Any request with a result size above this threshold will result in a server-side `RuntimeException`.

Use a smart synchronization strategy
On the other hand there are very rare occasions where you would actually need to fetch all of the clicks at once. Most of the times you will probably be interested in all clicks associated to a mailing or list. If you are intending to synchronize all clicks we strongly encourage you to use a pagination mechanism which is only fetching the clicks which were performed since the last synchronization. You still have to perform the initial synchronization of course. Be careful to keep the number of clicks fetched per request below a reasonable limit by applying appropriate filter conditions.

Close your row sets
One final word regarding `Inx_Api_DataAccess_ClickDataRowSet`: Be sure to close these resources once you have read all of the clicks and try to avoid keeping multiple `Inx_Api_DataAccess_ClickDataRowSets` alive simultaneously. The ID list on the server is stored until you close either the row set or the session. If you do neither of these, it will be discarded once the session is marked as inactive. Do *not* rely on this fact because the data will accumulate pretty fast depending on the amount of data you are synchronizing.

3.9.11. Inx_Api_Sending_SendingHistoryManager

The `Inx_Api_Sending_SendingHistoryManager` and the `Inx_Api_Sending_Sending` business object can be used to access data related to the sending of mailings. The following questions - and more - can be answered by this manager:

- When and to which recipients was a mailing sent?
- Did the mailing bounce?
- Did the recipient react on the mailing (opening/click)?
- How large was the sending and the average mail size?

Terminology note: In this chapter, mailings as they appear in the Inxmail Professional client are called "mailings", while the emails actually sent to recipients are called "mails".

The `Inx_Api_Sending_Sending` business object represents the sending of a particular mailing to a set of recipients. A sending is either triggered by an event (e.g. subscription, action, manual sending, etc.) or if the scheduled sending date is reached. While regular mailings are usually only sent once, trigger mailings may be sent an unlimited number of times.

Each sending consists of "individual sendings", one for each contacted recipient. These entries are a kind of protocol for the sending. They keep track of the contacted recipients, their reaction on the mail and the current status of the sending regarding this recipient.

To understand how these components work together it is helpful to understand how Inxmail Professional sends mailings. After a sending is triggered, a sending object is created. This object corresponds to the sending business object and keeps track of the state of the sending and - through an additional server call - grants access to some accumulated statistics. The next step is to personalize the mailing for each recipient who will be contacted. When the mailing is ready to be sent, the start date of the sending is set and the actual sending process begins. For each recipient of the sending an "individual sending" is created, keeping track of the state of the sending process and the reaction of the recipient. After all mails have been sent, the end date of the sending is set.

There are a number of different criteria by which sending objects can be retrieved. Mainly these are combinations of the mailing ID, the recipient ID and the date range. Additionaly, it is possible to find modified sendings which at the same time enables the pagination of sending data. The following events are considered as modifications:

- The sending was triggered (created)
- The sending was started
- The sending was finished
- A mail of the sending was sent to a recipient
- A recipient of the sending opened the mail
- A recipient of the sending clicked a link of the mail
- A recipient of the sending caused a bounce
- The mailing was deleted
- The sending protocol (individual sendings) was deleted

This list is not exhaustive.

The following methods can be used to retrieve sendings:

```
public function findSendingsByMailing( $iMailingId );
public function findSendingsByRecipient( $iRecipientId );
public function findSendingsByDate( $sStart = null, $sEnd = null );
public function findPastSendingsByMailing( $iMailingId, $sStart = null, $sEnd = null );
public function findPastSendingsByRecipient( $iRecipientId, $sStart = null, $sEnd = null );
public function findModifiedSendings( $sSince );
public function findLastSendingForMailing( $iMailingId );
public function findLastSendingForRecipient( $iRecipientId );
public function findLastSending();
```

The following snippet demonstrates how to retrieve all sendings for a mailing which were processed during the last 30 days:

```
$sStart = date('c', strtotime('-30 days'));

$oSendingHistoryManager = $oSession->getSendingHistoryManager();
$oSendings = $oSendingHistoryManager->findPastSendingsByMailing( $iMailingId, $sStart, null );
```

Apart from retrieving sending business objects, the `Inx_Api_Sending_SendingHistoryManager` may also be used to retrieve the next expected sending dates. Be aware that it is not guaranteed that a sending will be performed at the dates returned. If the sending process is triggered at a point of time when no recipients match the criteria or there are no recipients at all, there will be no actual sending. Also note, that theses dates do not specify the actual point in time at which the first mail is sent. As mentioned before, the mailing has to be prepared (personalized) for each recipient before the first mail is sent.

The following methods can be used to retrieve the expected future sending dates (of a mailing):

```
public function findNextSending( $iMailingId );
public function findFutureSendingsByMailing( $iMailingId, $sStart, $sEnd );
public function findFutureSendingsByDate( $sStart, $sEnd );
```

In addition, the `Inx_Api_Sending_SendingHistoryManager` allows simplified access to the reactions of single recipients. There are two kinds of these methods: Those which expect date parameters and those which do not. The difference is the following: The methods without date parameters only take into account the last sending of the mailing. The methods with date parameters take into account all sendings which were performed during the given time span. Passing in null dates here takes every sending of the mailing into account. Keep in mind that trigger mailings might be sent an arbitrary number of times.

The following methods can be used to retrieve the reactions of single recipients:

```
public function hasOpened( $iRecipientId, $iMailingId );
public function hasClicked( $iRecipientId, $iMailingId );
public function hasBounced( $iRecipientId, $iMailingId );
public function hasOpenedBetween( $iRecipientId, $iMailingId, $sStart, $sEnd );
public function hasClickedBetween( $iRecipientId, $iMailingId, $sStart, $sEnd );
public function hasBouncedBetween( $iRecipientId, $iMailingId, $sStart, $sEnd );
```

As mentioned before, the `Inx_Api_Sending_Sending` business object keeps track of the status of the whole sending. The following information can be retrieved:

- The ID of the sending
- The ID of the mailing to be sent
- The ID of the list containing the mailing to be sent
- The start date of the sending (after personalization)
- The end date of the sending
- The modification date of the sending
- The state of the sending
- The type of the mailing to be sent
- The total size of the sending in bytes (including all mails already sent)
- A boolean indicating whether the mailing was deleted
- A boolean indicating whether the protocol (individual sendings) was deleted
- The recipient reactions, including meta data if needed
- All clicks on links in the mailing of the sending

If the mailing associated with the sending still exists and is compatible with the `Inx_Api_General-Mailing_GeneralMailingManager` you can also retrieve a read-only view of the mailing as demonstrated in the following snippet:

```
$oSending = $oSession->getSendingHistoryManager()->findLastSending();
$oMailing = $oSending->findGeneralMailing();

if( $oMailing != null )
{
   $listId = $oMailing->getListContextId();
   $mailingId = $oMailing->getId();
   $mailingName = $oMailing->getName();
   $mailingSubject = $oMailing->getSubject();

   echo "$listId: $mailingId – $mailingName / $mailingSubject<br>";
}
```

In addition, the `Inx_Api_Sending_Sending` business object grants access to some accumulated statistics through the `getReportData` method which fetches a `Inx_Api_Sending_SendingReport` object. Be aware that this method performs an additional server call. The following information can be retrieved using the `Inx_Api_Sending_SendingReport` object:

- The number of recipients who opened the mail
- The number of recipients who clicked a link of the mailing
- The number of mails sent, including bounces
- The number of mails sent, excluding bounces
- The number of recipients who caused a bounce
- The number of mails which have not yet been sent
- The average size of the mails

There are several ways of retrieving recipient reactions. The easiest approach is to fetch the data as `Inx_Api_Sending_IndividualSendingRowSet`. This row set contains the recipient ID, the state of the sending to that recipient and boolean flags indicating whether the recipient opened the mail, clicked a link or caused a bounce.

If you need to access recipient meta data - column data and state - use a `Inx_Api_Sending_SendingRecipientRowSet`. This row set includes all the information accessible through the `Inx_Api_Sending_IndividualSendingRowSet` but also allows to retrieve recipient meta data.

If you need to modify the recipients of the sending but you do not need to consider their reactions, use a `Inx_Api_Recipient_RecipientRowSet` which is also available from the sending.

The following table depicts the functionality of the various methods:

	Reaction (single)	Reaction (bulk)	Meta data	Manipulation
hasOpened	X	-	-	-
hasClicked	X	-	-	-
hasBounced	X	-	-	-
findIndividualSendings	-	X	-	-
findClicks	-	X	X	-
findSendingRecipients	-	X	X	-
findRecipients	-	-	X	X

There is no direct way of accessing recipient reactions and at the same time manipulating recipient data. To do this you need a two-stages approach:

1. Collect the relevant recipient IDs using `findIndividualSendings()`

2. Call `Inx_Api_Recipient_RecipientContext->findByIds($aIds)` to manipulate these recipients

The following example demonstrates how to determine all recipients who opened the sent mail and set a date flag for these recipients:

```
$oSendingHistoryManager = $oSession->getSendingHistoryManager();
$oLastSending = $oSendingHistoryManager->findLastSendingForMailing( $iMailingId );
$oIndividualSendings = $oLastSending->findIndividualSendings();

$aRecipientIds = array();

while( $oIndividualSendings->next() )
{
  if( $oIndividualSendings->hasOpened() )
  {
    $aRecipientIds[] = $oIndividualSendings->getRecipientId();
  }
}

$oIndividualSendings->close();

$oRecipientContext = $oSession->createRecipientContext();
$oLastOpening = $oRecipientContext->getMetaData()->getUserAttribute( 'LastOpening' );
$oRecipients = $oRecipientContext->findByIds( $aRecipientIds );

$sNow = date( 'c' );

while( $oRecipients->next() )
{
  $oRecipients->updateDatetime( $oLastOpening, $sNow );
  $oRecipients->commitRowUpdate();
}

$oRecipients->close();
```

Performance Considerations

The sending data volume in Inxmail Professional can be rather huge. This is a factor you need to consider when using the `Inx_Api_Sending_SendingHistoryManager`. The large volume of data results from the fact that for each sending there is a record for each and every recipient who was supposed to be contacted, regardless of whether the mail could actually be delivered to that recipient or not.

Let's say the system sends one mailing per day to a recipient base of one million recipients. Taking into consideration that sending history data is usually stored for two years that makes 730 Sendings to one million recipients. That would amount to a total of 730 million records!

Scanning this amount of data naturally takes some time. That is why the `Inx_Api_Sending_SendingHistoryManager` offers a layered approach to accessing the relevant data. The less data you need, the faster the request will be.

The Sending BusinessObject

This implies that if you access more data you need to talk to the server more often. Because the additional server calls are transparent, it is not obvious that some of the methods on the `Inx_Api_Sending_Sending` BusinessObject actually do perform one. Depending on the size of your Inxmail application the time this call takes might be quite considerable. The following methods of the `Inx_Api_Sending_Sending` object perform a server call:

- `getReportData`
- `hasOpened`

- `hasClicked`
- `hasBounced`
- `findIndividualSendings`
- `findSendingRecipients`
- `findClicks`
- `findRecipients`
- `findGeneralMailing`

The time these server calls take varies greatly. The `has*` methods usually require just a few milliseconds even on installations as big as 500 million records. On the other hand, `getReportData` may take up to 10 seconds on such an installation. The `find*` methods might even take up to 15 seconds.

Regarding the `find*` methods there is also another aspect you need to take into consideration: pagination. As an `Inx_Api_Sending_IndividualSendingRowSet` might contain several million entries - keep in mind there will be one entry per recipient - it is impossible to fetch all the data at once. This would simply cause a timeout. As all row sets and result sets in the Inxmail Professional API, the row sets used in the sending history fetch data in chunks:

- `findIndividualSendings`: 1000 entries at once, per default
- `findSendingRecipients`: 500 entries at once, per default
- `findClicks`: 500 entries at once, per default
- `findRecipients`: 50 entries at once, per default
- `findGeneralMailing`: 50 entries at once, per default

As stated earlier in this chapter, the `Inx_Api_Sending_IndividualSendingRowSet` only contains sending states and recipient reactions; no recipient metadata.

The `Inx_Api_Sending_SendingRecipientRowSet` contains the same data plus any recipient attributes you specified. Make sure to use as few attributes as possible, the less attributes you fetch, the less time this call will require, including the calls performed during pagination of the row set.

Finally, the `Inx_Api_Recipient_RecipientRowSet` includes the complete recipient record. Depending on the Inxmail application this might be several thousands of attributes. That is why the chunk size is so small for `Inx_Api_Recipient_RecipientRowSet`s.

As you can see, the more data a method fetches, the smaller the chunk size gets, which is quite natural.

The SendingHistoryManager
Most of the methods in the `Inx_Api_Sending_SendingHistoryManager` are quite fast, even in large installations of Inxmail Professional. The following methods usually return in a matter of milliseconds, again depending on the scale of the target system:

- `get`
- `selectAll()`
- `findSendingsByMailing`
- `findSendingsByRecipient`

- `findSendingsByDate`
- `findPastSendingsByMailing`
- `findPastSendingsByRecipient`
- `findModifiedSendings`
- `findLastSendingForMailing`
- `findLastSendingForRecipient`
- `findLastSending`
- `hasOpened`
- `hasOpenedBetween`
- `hasClicked`
- `hasClickedBetween`
- `hasBounced`
- `hasBouncedBetween`
- `findNextSending`

There are two methods, however, which may take considerably more time:

- `findFutureSendingsByMailing`
- `findFutureSendingsByDate`

The performance of these two strongly correlates with the date range you specify. Small ranges will perform quite well. If you use ranges of up to, say, one year, that will take a significant amount of time; given you have some trigger mailings which are triggered on a regular basis. The same is true for sequence mailings.

If you can settle for the `findFutureSendingsByMailing` method instead of the `findFutureSendingsByDate` method, this is definitely something to consider because `findFutureSendingsByDate` has to check each and every scheduled mailing, trigger mailing, sequence mailing and split-test mailing. Depending on the size of the installation this might be quite a lot. Restraining this request to a single mailing in a preferably narrow time span will significantly increase the performance.

3.9.12. Inx_Api_Action_ActionManager

The action manager can be used to search, create and modify actions. Creating actions is done using the `createAction` method. But before new actions can be commited, the action type has to be set which specified the event which triggers the action.
Following action types do not need a list context to be specified, since they are system wide:

- `EVENT_TYPE_CLICK` - A link in an email is clicked.
- `EVENT_TYPE_HARD_BOUNCE` - Hard bounce mail received.
- `EVENT_TYPE_SOFT_BOUNCE` - Soft bounce mail received.
- `EVENT_TYPE_UNKNOWN_BOUNCE` - Unknown mail detected through the bounce mailbox.
- `EVENT_TYPE_AUTO_RESPONDER_BOUNCE` - Auto-responder mail received through the bounce mailbox.

- `EVENT_TYPE_AUTO_RESPONDER_REPLY` - Auto-responder mail received through the normal mailbox.
- `EVENT_TYPE_FLAME_REPLY` - Flame mail received through the normal mailbox.
- `EVENT_TYPE_FLAME_REPLY` - Unknown mail detected through the bounce mailbox.

Following event types need a list context (`Inx_Api_List_StandardListContext` or `Inx_Api_List_FilterListContext`) specified:

- `EVENT_TYPE_NEWSLETTER_SENT` - A newsletter was sent.
- `EVENT_TYPE_SINGLE_MAIL_SENT` - A single mail was sent.
- `EVENT_TYPE_SUBSCRIBE` - A recipient was successfully subscribed.
- `EVENT_TYPE_UNSUBSCRIBE` - A recipient was successfully unsubscribed.

If an action is triggered, it executes predefined commands. These commands are build by a `Inx_Api_Action_CommandFactory`, which is returned from the `getCommandFactory` method of the `Inx_Api_Action_ActionManager`. These factory methods are available:

```
public function createDeleteRecipientCmd();
public function createSetValueCmd( $iAttributeId,
                                   $sExpression );
public function createSetAbsoluteValueCmd( $iAttributeId,
                                   $sAbsoluteValue );
public function createSetRelativeValueCmd( $iAttributeId,
                                   $sRelativeValue );
public function createSubscriptionCmd( $iListContextId,
                                   $blSubscriptionProcessingEnabled );
public function createUnsubscriptionCmd( $iListContextId,
                                   $blSnsubscriptionProcessingEnabled );
public function createUnsubscribeAllCmd();
public function createSendLastNewsletterCmd( $iListContextId );
public function createSendMailCmd( $iListContextId, $iMailingId );
public function createSendActionMailCmd( $iListContextId, $iActionMailingId );
```

Creating an Action

Following example creates a new "On Click" action, which sets the current date into the profile attribute `lastClickAttr`, and increments an integer counter in the attribute `clickCountAttr`.

```
$oListContextManager = $oSession->getListContextManager();
$oListContext = $oListContextManager->findByName(
                        Inx_Api_List_SystemListContext::NAME);

$oActionManager = $oSession->getActionManager();
$oAction = $oActionManager->createAction($oListContext);
$oAction->updateEventType(Inx_Api_Action_Action::EVENT_TYPE_CLICK);
$oAction->updateName("Click?Registry") ;

$oFactory = $oActionManager->getCommandFactory();
$cmds = array();
$cmds[0] = $oFactory->createSetValueCmd ( $lastClickAttr , "=Date()" ) ;
$cmds[1] = $oFactory->createSetRelativeValueCmd ( $clickCountAttr , 1 ) ;
$oAction->updateCommands($cmds);
$oAction->commitUpdate();
```

3.9.13. Inx_Api_Blacklist_BlacklistManager

Blacklist rules, managed by the Blacklist Manager, block email addresses matched by these rules from Inxmail. These addresses can not find their way into Inxmail, neither by import nor by subscription or in other ways.

You activate the blacklist feature on the `Inx_Api_List_SystemListContext`:

```
$oListContextManager = $oSession->getListContextManager();
$oSystemListContext =
    $oListContextManager->findByName(Inx_Api_List_SystemListContext::NAME);
$oSystemListContext->enableFeature(Inx_Api_Features::BLACKLIST_FEATURE_ID);
```

In the blacklist, you can lock out individual addresses or whole complete address ranges. Examples:

- `name@firm.com` - The address 'name@firma.com' is blocked
- `*firm.com` - All personnel of this firm is locked out
- `*.tv` - No addresses from Tavaluga
- `spam*` - All addresses beginning with 'spam' are blocked
- `martin@*` - All Martins are blocked

```
public function createBlacklistEntry();
public function findByPattern( $sPattern );
public function selectAll( $orderAttribute, $orderType );
```

Adding new Rules

To add new rules, create a blacklist entry and update its pattern.

```
$oBlacklistManager = $oSession->getBlacklistManager();
$oBlacklistEntry = $oBlacklistManager->createBlacklistEntry();
$oBlacklistEntry->updatePattern("*@spamcop.com");
$oBlacklistEntry->updateDescription("No addresses from SpamCop");
$oBlacklistEntry->commitUpdate();
//Now, all SpamCop addresses have been removed .
echo "Deleted : " . $oBlacklistEntry->getHitCount();
```

Searching entries

Since Inxmail Professional 3.7 you can search for blacklist entries. You can use the following methods to search in the blacklist. For example you can search for all modified or created entries between to dates.

Note: Only changes in the description or pattern updates the modification date of the blacklist entry.

```
public function selectAfter( $searchDate );
public function selectBefore( $searchDate );
public function selectBetween( $startDate, $stopDate );
```

The following example shows the retrieving of blacklist entries for 24 hours.

```
$blm = $session->getBlackListManager();
$rs = $blm->selectBetween('2008-01-01T00:00:00.000Z','2008-01-02T00:00:00.000Z');
for($i = 0; $i < $rs->size(); $i++)
{
...
}
$rs->close();
```

3.9.14. Managing Resources

Attachments used in mailings are "resources". Using the `Inx_Api_Resource_ResourceManager`, these resources can be upload to and download from the Inxmail server. Resources can be bound

to mailing lists or mailings, which means they are not visible outside these bounds, and will be removed with their mailing list or mailing.

```
$oResourceManager = $oSession->getResourceManager();
$in = fopen( "/images/logo.gif", 'rb' );
$oResource = $oResourceManager->upload ( null, "logo.gif", $in);
fclose($in);
```

Inxmail assigns to the so uploaded resource a unique identifier. To attach a resource to a mailing, add the `attach` tag to the mail body:

```
$s = '[%attach('. $oResource->getId() . ');' . $oResource->getName() . ']';
```

This results in a string like `[%attach(42); logo.gif]`. To locate existing resources, use the `select` methods of the `Inx_Api_Resource_ResourceManager`.

3.9.15. Inx_Api_Bounce_BounceManager

Since Inxmail Professional 3.7 it is possible to activate VERP (Variable envelope return path) in the mailserver settings. With activated VERP, all bounce objects containing a mailing id, list id and recipient id, if they are available. Also you can retrieve the bounce mailing as input stream. With the Inxmail API 1.4.3 we introduce a bounce handling for managing the bounces over the Inxmail API. This makes it easy to synchronise the bounces to a third party system.

Note:

- Every result set can include bounces which occurred while testing the mailing (sending to test recipients).
- The bounce count in the sending info can be different from the size of the result set. Because bounces can be deleted.

The `Inx_Api_Bounce_BounceManager` contains the methods for retrieving bounce objects.

```
interface Inx_Api_Bounce_BounceManager extends Inx_Api_BOManager
{
  public function selectBefore( $searchDate, Inx_Api_Recipient_RecipientContext $oRc = null,array $aAttrs = null );
  public function selectAfter( $searchDate, Inx_Api_Recipient_RecipientContext $oRc = null,array $aAttrs = null );
  public function selectBetween( $startDate, $stopDate, Inx_Api_Recipient_RecipientContext $oRc = null,array $aAttrs =
      null );
  public function selectByMailingId( $mailingId, Inx_Api_Recipient_RecipientContext $oRc = null,array $aAttrs = null );
  public function selectByListId( $listId, Inx_Api_Recipient_RecipientContext $oRc = null,array $aAttrs = null );
  public function selectAllBounces(Inx_Api_Recipient_RecipientContext $oRc = null, array $aAttrs = null);
  public function createQuery(Inx_Api_Recipient_RecipientContext $oRc = null, array $aAttrs = null);
}
```

Following bounce categories are defined:

- `CATEGORY_HARD_BOUNCE` - Incoming mail is categorized as hard bounce.
- `CATEGORY_SOFT_BOUNCE` - Incoming mail is categorized as soft bounce.
- `CATEGORY_AUTO_RESPONDER_BOUNCE` - Incoming mail is categorized as auto responder bounce (since Inxmail Professional API 1.12.1).
- `CATEGORY_SPAM_BOUNCE` - Incoming mail is categorized as spam bounce (since Inxmail Professional API 1.12.1).
- `CATEGORY_UNKNOWN_BOUNCE` - Incoming mail can not be categorized as one of the above categories.

The following sample shows the retrieval of bounces for a given mailing.

```
$mailingId = ...;
$bm = $s->getBounceManager();
$rs = $bm->selectByMailingId( $mailingId );
for($i = 0; i < $rs->size(); $i++)
{
...
}
$rs->close();
```

Fluent interface for bounce queries
In Inxmail Professional API 1.12.1, a new fluent interface for retrieving bounces was introduced. The basic idea is to simply create a query object and combine the available filters as you need instead of figuring out which method offers the appropriate set of filters. This allows you to create complex queries, while the fluent interface keeps the syntax as concise as possible, thus producing more readable and maintainable code.

Using the new fluent query interface, you can now filter the bounces by date, list, mailing and bounce category combined in one query.

The following sample demonstrates a common bounce query: retrieving all bounces which were received during the last 24 hours in a particular list. Note that the last two lines show the actual query.

```
$oRecipientContext = $oSession->createRecipientContext();
$aAttrs = array( $oRecipientContext->getMetaData()->getEmailAttribute() );
$sStart = date( 'c', strtotime( '-1 day' ) );
$aListIds = array( 3 );

$oBounceQuery = $oSession->getBounceManager()->createQuery( $oRecipientContext, $aAttrs );
$oBOResultSet = $oBounceQuery->listIds( $aListIds )->after( $sStart )->executeQuery();
```

3.9.16. Inx_Api_Inbox_InboxManager

Of course bounce notifications aren't the only messages the Inxmail Professional server can handle. The server will also manage responses sent by customers. Since version 1.9.0 of the Inxmail Professional API it is possible to manage these inbox messages using the Inx_Api_Inbox_InboxManager. This manager is organized pretty much the same way as the `Inx_Api_Bounce_BounceManager`, though the inbox message object contains less information due to technical restrictions with email replies.

Note: It is generally possible to retrieve recipient attributes for the sender of an inbox message if the sender is known to Inxmail Professional as a recipient. However, if the sender address is unknown, the recipient status will be `RECIPIENT_STATE_UNKNOWN` and fetching recipient attributes will raise an `Inx_Api_UnknownRecipientException`.

The `Inx_Api_Inbox_InboxManager` defines the following methods:

```
interface Inx_Api_Inbox_InboxManager extends Inx_Api_BOManager
{
  public function selectBefore( $sSearchDate, Inx_Api_Recipient_RecipientContext $rc, $aAttributes );
  public function selectAfter( $sSearchDate, Inx_Api_Recipient_RecipientContext $rc, $aAttributes );
  public function selectBetween( $sStartDate, $sStopDate, Inx_Api_Recipient_RecipientContext $rc, $aAttributes );
  public function selectAllInboxMessages( Inx_Api_Recipient_RecipientContext $rc = null, $aAttributes = null );
}
```

Following inbox message categories are defined:

- `CATEGORY_AUTO_RESPONDER` - Incoming mail is categorized as auto responder mail.
- `CATEGORY_FLAME` - Incoming mail is categorized as flame message with aggressive content and/or strong language.

- `CATEGORY_SPAM` - Incoming mail is categorized as undesirable by spam/virus checking software.
- `CATEGORY_UNCATEGORIZED` - Incoming mail is an ordinary mail which does not match a specific category.
- `CATEGORY_UNKNOWN` - The category of the incoming mail is unknown. This indicates a version mismatch of server and API.

The following sample shows the retrieval of inbox messages which were received since yesterday:

```
$sYesterday = date('c', strtotime("-1 day"));

$oRecipientContext = $oSession->createRecipientContext();
$oRecipientMetaData = $oRecipientContext->getMetaData();
$firstname = $oRecipientMetaData->getUserAttribute( "Firstname" );
$lastname = $oRecipientMetaData->getUserAttribute( "Lastname" );

$oInboxManager = $oSession->getInboxManager();
$oBOResultSet = $oInboxManager->selectAfter( $sYesterday, $oRecipientContext, array( $firstname, $lastname ) );

for( $i = 0; $i < $oBOResultSet->size(); $i++ )
{
  $oInboxMessage = $oBOResultSet->get( $i );

  echo 'Subject: ' . $oInboxMessage->getSubject() . '<br>';
  echo 'Received at: ' . $oInboxMessage->getReceptionDate() . '<br>';

  if( $oInboxMessage->getRecipientState() == Inx_Api_Inbox_InboxMessage::RECIPIENT_STATE_EXISTENT )
  {
    echo 'Sent by: ' . $oInboxMessage->getString( $firstname ) . ' '
        . $oInboxMessage->getString( $lastname ) . '<br>';
  }
  else
  {
    echo 'Sent by: Unknown<br>';
  }
}
```

The code in the sample above prints out some basic information about the message: the subject, the date of reception and the name of the sender. Note that the recipient attributes are only fetched if the sender was recognized by Inxmail Professional and was not deleted.

3.9.17. Test profiles

Since Inxmail Professional 3.8 it is possible to create test recipients. With the Inxmail Professional API 1.6.0 it is possible to access the test profiles from the API. Test recipients are similar to the normal recipients, so the handling in the Inxmail Professional API is similar to the *Inx_Api_Recipient_RecipientContext*. The following sample shows creating a new test recipient for a list.

```
$lc = ...;
$trc = $session->createTestRecipientContext();
$rc = $session->createRecipientContext();
$rmd = $rc->getMetaData();
$trs = $trc->createRowSet($lc);
$trs->moveToInsertRow();
$trs->updateString( $rmd->getEmailAttribute(), "test@invalid.invalid" );
$trs->updateName("Test profile created by API");
$trs->commitRowUpdate();
$trs->close();
$rc->close();
```

3.9.18. Inx_Api_Webpage_WebpageManager

Web pages are mainly used as landing pages for the subscription and unsubscription process, though they can be used for many other purposes as well. Since version 1.9.0 of the Inxmail Professional API it is possible to retrieve information about the configured web pages using the `Inx_Api_Webpage_WebpageManager`.
The manager offers several select methods which can be used to search for specific web pages. The most important filter is the web page type which can be JSP (dynamic) or HTML form (static). Another filter is used to retrieve web pages by their sub type. The sub type is a string which is used internally by the Inxmail Professional server to define the usage of the web page. For example, subscription landing pages have the sub type 'subscription'.
The following example illustrates how to retrieve all subscription JSPs and print out their names and URLs:

```
$oWebpageManager = $oSession->getWebpageManager();
$oBOResultSet = $oWebpageManager->selectJspsBySubType( 'subscription' );

for( $i = 0; $i < $oBOResultSet->size(); $i++ )
{
  $oWebpage = $oBOResultSet->get( $i );

  echo 'Name: ' . $oWebpage->getName() . '<br>';
  echo 'URL: ' . $oWebpage->getServerUrl() . '<br>';
}
```

3.9.19. Retrieving Reports

Reports need to be configured before they can be generated. This is done with the `Inx_Api_Reporting_ReportRequest` object. The asynchronous report generation process state is controlled with a `Inx_Api_Reporting_ReportTicket`. For each report to generate, such a ticket has to be acquired. As soon as the report has been generated, it can be downloaded with the `Inx_Api_Reporting_DownloadableResult`.
Following example creates a "System Domain Distribution", showing not more than 20 domains and outputting as HTML. All texts will be in German (de) language (for a list of available reports and their parameters see appendix A.):

```
$oReportRequest = new Inx_Api_Reporting_ReportRequest("SystemDomainDistribution",
          Inx_Api_Reporting_ReportRequest::OUTPUT_FORMAT_HTML,
          "de" ,'Europe/Berlin' ) ;
$oReportRequest->putParameter ("limit", "20");
```

Using the request, the report generation can be requested. As soon as the report is available, the report ticket will return a valid "downloadable result":

```
$oReportTicket = $oSession->getReportEngine()->generate($oReportRequest, false) ;
$oDownloadableResult = $oReportTicket->fetchDownloadableResult();

while ( $oDownloadableResult == null )
{
    // Waiting for the report to finish ...
    sleep(3);
    $oDownloadableResult = $oReportTicket->fetchDownloadableResult();
}

$sOutputFile = "SystemDomainDistribution.pdf";
download( $oResult->getInputStream(), $sOutputFile );

if( $oReportTicket != null )
{
  $oReportTicket->close();
}
```

Generated reports are cached on the Inxmail server. The default time in cache is set to two hours. If `ignoreCache` parameter of the report engine's generate method is `true`, the server cache will be

ignored and reports always regenerated.
Following code handles the download of reports. Reports as HTML and CSV format will be transferred as ZIPped file, since they normally contain more than one file:

```
function download($inputStream, $sFileName)
{
   $handle = fopen($sFileName, 'w+b');
   while (($ch = $inputStream->read()) != -1)
   {
      fwrite($handle, $ch);
   }

      $inputStream->close();
      fclose($handle);
}
```

A. Reports Reference

A.1. Catalogues

Catalogues are the first pages displayed in Inxmail Client's the Report agent ("home"), presenting a list of available reports. There are three of them, one for the system list, one for mailing lists, and one for mailings.

Note: The reports are not part of the Inxmail API, they can change on every release of Inxmail Professional!

Internal names:
List Reports - ListReportsCatalog
Mailing Reports - MailingReportsCatalog
General Reports - SystemReportsCatalog

A.2. Bounce Reports

A.2.1. Broken down by (top-level) domain

Internal name: BounceTypesByDomain, BounceTypesByToplevelDomain

Parameter	Data type	Description
begin	long	Start date of report
end	long	End date of report
count	integer	Number of days in the past from now
limit	integer	Number of rows in result

Internal name: BounceTypesByDomainByList, BounceTypesByToplevelDomainByList

Parameter	Data type	Description
listid	integer	List context identifier
begin	long	Start date of report
end	long	End date of report
count	integer	Number of days in the past from now
limit	integer	Number of rows in result

Internal name: BounceTypesByDomainByMailing, BounceTypesByToplevelDomainByMailing

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier
type	integer	Report mailing type
begin	long	Start date of report
end	long	End date of report
count	integer	Number of days in the past from now
limit	integer	Number of rows in result

A.2.2. Development over time

Internal name: IncomingMailDetails

Parameter	Data type	Description
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

Internal name: IncomingMailDetailsByList

Parameter	Data type	Description
listid	integer	List context identifier
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

Internal name: IncomingMailDetailsByMailing

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier
type	integer	Report mailing type
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

A.2.3. Bounces and replies by Domain

Internal name: IncomingMailDetailsForDomain

Parameter	Data type	Description
begin	long	Start date of report
end	long	End date of report
count	integer	Number of days in the past from now
interval	string	Time interval type (hour,day,week,month)
domain	string	Domain name

A.2.4. Broken down by top 5 domains over time

Internal name: TimedIncomingMailByDomain

Parameter	Data type	Description
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

Internal name: TimedIncomingMailByDomainByList

Parameter	Data type	Description
listid	integer	List context identifier
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

Internal name: TimedIncomingMailByDomainByMailing

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier
type	integer	Report mailing type
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

A.2.5. Broken down by top-level domains over time

Internal name: TimedIncomingMailByTopLevelDomain

Parameter	Data type	Description
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

Internal name: TimedIncomingMailByTopLevelDomainByList

Parameter	Data type	Description
listid	integer	List context identifier
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

Internal name: TimedIncomingMailByTopLevelDomainByMailing

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier
type	integer	Report mailing type
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

A.3. Mailing Reports

A.3.1. Clicks related to weekday and hour

Internal name: ClickOverviewTimeUnit

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier

A.3.2. Clicks related to individual links

Internal name: ClickReactionLink

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier

A.3.3. Click development over time

Internal name: ClickReactionTimeResponse

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier
interval	string	Time interval type (hour,day,week,month)
count	integer	Number of intervals since dispatch date

A.3.4. Most important key data of mailing

Internal name: MailingDetailOverview, SplitTestMailingDetailOverview

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier

Internal name: TriggerMailingDetailOverview, SubscriptionWelcomeMailingDetailOverview

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier
type	integer	Report mailing type

A.3.5. Sendings overview

Internal name: TriggerMailingSendingsOverview

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier
type	integer	Report mailing type
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now

Internal name: SubscriptionWelcomeSendings

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier
type	integer	Report mailing type

A.3.6. Split test analysis

Internal name: SplitTestResult

Parameter	Data type	Description
listid	integer	List context identifier
splittestid	integer	Split test identifier

A.3.7. E-mail clients used

Internal name: UserAgent

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier
filterid	integer	Targetgroup identifier
count	integer	Number of intervals in the past from now

A.4. Recipient Demographics

A.4.1. Analysis of recipient data

Internal name: SystemAttributeDistribution, AttributeDistribution

Parameter	Data type	Description
listid	integer	List context identifier (only AttributeDistribution)
limit	integer	Number of rows in result
attrid	integer	Attribute id

A.4.2. Domain distribution

Internal name: SystemDomainDistribution, DomainDistribution

Parameter	Data type	Description
listid	integer	List context identifier (only DomainDistribution)
limit	integer	Number of rows in result

A.4.3. Top-level domain distribution

Internal name: SystemTopLevelDomainDistribution, TopLevelDomainDistribution

Parameter	Data type	Description
listid	integer	List context identifier (only TopLevelDistribution)
limit	integer	Number of rows in result

A.5. List Reports

A.5.1. Most important key data of a list

Internal name: ListOverview

Parameter	Data type	Description
listid	integer	List context identifier
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

A.5.2. Send overview

Internal name: ListSentOverview, SystemSentOverview

Parameter	Data type	Description
listid	integer	List context identifier (only ListSentOverview)
begin	long	Start date of report
end	long	End date of report
count	integer	Number of days in the past from now

A.5.3. Analysis of transport frequency

Internal name: SendFrequency

Parameter	Data type	Description
listid	integer	List context identifier
begin	long	Start date of report
end	long	End date of report
count	integer	Number of days in the past from now

A.5.4. Evolution over time

Internal name: SubscriptionTimeResponse

Parameter	Data type	Description
listid	integer	List context identifier
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

A.5.5. Related to weekday and daytime

Internal name: SubscriptionTimeUnit

Parameter	Data type	Description
listid	integer	List context identifier
begin	long	Start date of report
end	long	End date of report
count	integer	Number of days in the past from now

A.5.6. Comparison of mailings in current list

Internal name: CompareMailingDetailOverview

Parameter	Data type	Description
listid	integer	List context identifier
mailingids	string	List of mailing ids, Note: use # as separator!
interval	string	Time interval type (hour,day,week,month)
count	integer	Number of intervals in the past from noww

A.5.7. Target group comparison of current mailing

Internal name: TargetGroupClickReport

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier
selectedLinkIds	string	List of link ids, Note: use # as separator!
targetGroupIds	string	List of targetgroup ids, Note: use # as separator!
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

A.5.8. E-mail clients used

Internal name: UserAgentByList

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier
filterid	integer	Targetgroup identifier
count	integer	Number of intervals in the past from now

A.6. Administrative Reports

A.6.1. Mail server

Internal name: MailServer

Parameter	Data type	Description
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

A.6.2. Analysis of sending mail server (SMTP)/(POP3)

Internal name: MailServerDetail

Parameter	Data type	Description
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)
server	string	Name of the mail server
type	string	Type of the mail server (pop3,smtp)

A.7. General Reports

A.7.1. Overview of the most important key data of all lists

Internal name: SystemOverview

Parameter	Data type	Description
begin	long	Start date of report
end	long	End date of report
count	integer	Number of intervals in the past from now
interval	string	Time interval type (hour,day,week,month)

A.7.2. E-mail volume

Internal name: SendRevenue

Parameter	Data type	Description
begin	long	Start date of report
end	long	End date of report

A.7.3. E-mail clients used

Internal name: UserAgentSystem

Parameter	Data type	Description
listid	integer	List context identifier
mailingid	integer	Mailing identifier
filterid	integer	Targetgroup identifier
count	integer	Number of intervals in the past from now

B. Support and Copyright

Inxmail is registered trademark of Inxmail GmbH, Freiburg.
If you have any problems please contact support@inxmail.com.

Acknowledgment This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

inxmail PROFESSIONAL 4

Inxmail Professional Manual

Imprint

Publisher: Inxmail GmbH
Address: Wentzingerstr. 17, 79106 Freiburg
Phone: +49 761 296979-0
Fax: +49 761 296979-9
Email: info@inxmail.com
Web: www.inxmail.com

Date: 07/2015
Author: Stefan Biermann, Christian Gerteis

Founded in 1999, Inxmail is an email marketing expert, providing a solution, namely Inxmail Professional, for professional online marketers and agencies. Inxmail Professional is particularly popular among big groups and agencies due to its high level of security and outstanding system stability. For professional users, Inxmail acts as a technical facilitator that allows them to realise their creative ideas in digital dialogue marketing. The particular strengths of Inxmail Professional include its superior performance, unbeatable flexibility and its wealth of professional functions.