



# **SAS Marketing Automation 4.4 Integration Utilities User Guide**

SAS Institute  
Solutions, Analytics and Strategy R&D Division



## Table of Contents

<b>Table of Contents</b> .....	<b>2</b>
<b>Document History</b> .....	<b>4</b>
<b>Introduction</b> .....	<b>5</b>
<b>Document Overview</b> .....	<b>5</b>
<b>Section 1. What's New?</b> .....	<b>6</b>
<b>Section 2. Deployment Overview</b> .....	<b>7</b>
Typical Installation of the SAS MA Integration Utilities.....	7
JNDI Properties File.....	8
Manually Deployed Jar Files.....	8
Additional Jars required for WebLogic®.....	9
Additional Jars required for WebSphere®.....	9
File I/O permissions .....	9
Security Model .....	9
Intended Usage.....	10
Process Models .....	10
Command Line Interface (CLI) .....	11
Running the Command Line Interface.....	11
XML File Naming Convention .....	12
<b>Section 3. XML Schema used by the Integration Utilities</b> .....	<b>13</b>
Document Template Definition (DTD) .....	13
XML Schema for Campaign Objects .....	13
CampaignDO .....	14
Folder.....	14
UserRoles .....	15
Occurrences.....	15
Definition .....	15
CheckListItemDO.....	16
FieldInfoDO.....	16
StatusDO .....	17
Schedule.....	18
NodeDefinitionId Values .....	18
<b>Section 4. Using the Extract Utility (sasmaextract)</b> .....	<b>19</b>
Using Search Criteria (Operator "=") .....	19
Using Regular Expressions (Operator "**").....	19
Combining Operators and Multiple Properties.....	20
Obtaining Explicit Detail (detail="Explicit" attribute).....	20
Performance versus Detail .....	21
Search Criteria Using Folder Details .....	21
Searching folders within a particular business context.....	21
Extract Schemas and Searchable Objects .....	22
Overriding the Extract Output Location .....	24
Overriding the XSLT input and output designations.....	25
Extracting Definitions from a particular Business Context.....	25
Extracted Metadata Response XML.....	25
Formatted XML Output .....	26
Logical Units and Object References .....	26
Results Summary .....	26
System Errors Encountered During Processing .....	27
Business Context and Folder Details .....	27
<b>Section 5. Using the Import Utility (sasmaintport)</b> .....	<b>28</b>
Creating an Import Request XML file .....	28
Recovery Level.....	28
Import Mode.....	29
Example Import Request XML.....	29



Redundant Data from the Extract Process .....	30
Importing Definitions to a particular Business Context .....	30
<b>Section 6. Using the Campaign Metadata Promotion Utility (sasmapromote) .....</b>	<b>31</b>
Campaign Metadata Promotion .....	31
Creating a Promotion Request .....	31
Example Promote Request .....	32
Setting the Target Environment and Business Context .....	32
Target Environment .....	32
Target Business Context .....	33
Using XSLT with the Promotion Utility .....	33
<b>Section 7. Log Files Created by the Utilities .....</b>	<b>34</b>
<b>Section 8. Using XML Style-Sheet Transformations (XSLT).....</b>	<b>35</b>
Programming in XSLT .....	35
Example: Using XSLT for Promotion Requests.....	35
Campaign Promotion Spreadsheet .....	35
Saving the spreadsheet as XML.....	37
XML Namespace .....	38
Creating the Transformation Style-Sheet .....	38
Processing the Work-Book Section of the Spreadsheet XML .....	39
Processing the Work-Sheet Section of the Spreadsheet XML.....	39
Processing of the Target Work-Sheet .....	40
Processing of the Campaigns Work-Sheet.....	42
Using the Style-Sheet as input to the Promotion Utility .....	43
Using XSLT with the Extract and Import Utilities .....	44
<b>Section 9. Using the Public Java API.....</b>	<b>45</b>
Process Model .....	45
Output Attribute for Extract Requests .....	45
API Jar File .....	45
Using the API to Submit a Request .....	46
Public Classes of the Java API.....	46
Metadata Gateway Factory Class .....	47
Class Methods .....	48
getConnection(MAConnectionDetails, boolean).....	48
getConnection(MAConnectionDetails) .....	48
hasValidConnection(String) .....	48
hasValidConnection(MAConnectionDetails).....	49
clearCaches .....	49
getDefaultContextName .....	49
Connection Details Class.....	49
Class Methods .....	50
getUser .....	50
getPass .....	50
getDomain.....	50
getBusinessContextName .....	50
getNamingProvider .....	50
Metadata gateway Connection class .....	50
Class Methods .....	52
Submit(String) .....	52
Submit(String, String, String) .....	53
Transform(String, String) .....	54
transform(String, String, String) .....	54
close.....	54
logon .....	55
logOff .....	55
getUser .....	55
setAppServerType(String) .....	55
setAppServerType(String, boolean) .....	55
getKeyName .....	56
Closing Connection Objects .....	56



Supporting Classes.....	56
Logging Class .....	56
Extract Exception Class.....	56
<b>Appendix A1 – Example batch (.bat) scripts for Windows .....</b>	<b>57</b>
Path setting in Windows Command Scripts for Additional Jars and SAS Private JRE .....	57
Increasing Memory .....	57
<b>Appendix A2.1 - Increasing Memory Available to the Utilities .....</b>	<b>58</b>
<b>Appendix A2.2 – Setting the system default locale used by the JVM .....</b>	<b>59</b>
<b>Appendix A3 – Optional Configuration File (auxdataio.properties) .....</b>	<b>60</b>
Logging Properties.....	60
XML Properties .....	60
Process Control Properties.....	60
<b>Appendix B1 – Example Campaign Promotion Spreadsheet XML .....</b>	<b>61</b>
<b>Appendix B2 – Campaign Promotion XSLT Style-Sheet.....</b>	<b>63</b>
<b>Appendix C – Frequently Asked Questions and Known Issues .....</b>	<b>65</b>
Frequently asked Questions (FAQ's) .....	65
Known Issues.....	65

## Document History

Version	Date	Author	Comment
1.0	9 <sup>th</sup> September 2004	Dr C.G. Statham	First Draft
1.1	1 <sup>st</sup> October 2004	Dr C.G. Statham	First Draft for Review
1.2	8 <sup>th</sup> October 2004	Dr C.G. Statham	First Internal Release
2.0	17 <sup>th</sup> August 2005	Dr C.G. Statham	MA4.3 Update
3.0	1 <sup>st</sup> September 2006	Dr C.G. Statham	MA4.4 Update



## ***Introduction***

This document describes the SAS Marketing Automation Integration Utilities that allow for in-house and/or 3<sup>rd</sup> party data-level integration of other systems with the MA Solution. Users of these utilities will typically be SAS Consultants or 3<sup>rd</sup> party integration specialists who have a good understanding of the MA architecture.

This document is provided as a user guide to help facilitate setting up of the utilities and their use within integration projects. The document consists of 7 sections (chapters) that are described in the following overview:

## ***Document Overview***

The first section of this document describes what's new in the integration utilities for this release.

The second section of this document describes the deployment of the integration utilities and how they fit within the Marketing Automation Solution. It explains the intended usage of the utilities and describes the typical installation.

The third section of this document describes some of the XML schema used with the integration utilities. This section focuses on the schema that represents and extract campaign metadata object.

The fourth section of this document describes how to use the utilities to extract MA meta-data from a working SAS meta-data repository. It explains how to launch the extract utility and how to create a meta-data extract request using an XML document.

Section five of this document describes how to use the utilities to import new MA meta-data into a working SAS meta-data repository. It explains how to launch the import utility and how to create a meta-data extract request using an XML document.

Section six of this document explains how to use the utilities to perform campaign metadata promotion. This new feature allows campaigns to be moved between development, testing and production environments.

Section seven of this document describes the log files that are created by the utilities that contain the details of any errors and/or process messages.

Section eight of this document shows how XML Style-Sheet Transformations (XSLT) can be used as part of the import and extracts process to provide alternate XML schemas and results.

Section nine of this document describes the Java API that is provided by the Integration Utilities. It shows working examples and explains how this API can be used in conjunction with the SAS event broker to provide a web based service oriented architecture (SOA) for integration of MA with third party products.

Appendices at the end of this document also provide additional information and code samples. Code samples and XML files can also be obtained from the ZIP file (CodeSamples.zip) that accompanies this document. A PDF version of this document and the zip file are available from the SAS MA consultant's web-site. Please contact SAS Technical Support for further information.

## Section 1. What's New?

A number of new features have been added to the Integration Utilities. These were predominantly included in the MA4.3 release of the solution but have also been added to in the MA4.4 release with the introduction and support for extract, import and promotion of export definitions. In MA4.3, the import capabilities were greatly extended and in the latest release of the solution it is now possible to fully extract and fully import campaigns, communications, diagrams and campaign/communication definitions. For example, a complete campaign (including any associated communications, diagram, definitions, user defined fields, etc) can be extracted to an XML file and then later restored to the metadata repository in just a few simple steps. This gives the ability to store or 'archive' campaigns in a simple file-based repository. In addition, a new 'campaign metadata promotion' facility also allows these campaigns to be ported across multiple metadata repositories (e.g., development, test and production systems). Several other features have also been added. An outline of just some of these new features is given below:

- **New Import Modes**
  - New – Creates New Campaigns, Definitions, etc.
  - Amend – Changes attributes of an existing Campaign, Definition, etc.
  - Replace – Completely replaces an existing Campaign, Definition, etc.
  - Delete – Removes an existing Campaign, Definition, etc.
  
- **Greater coverage of MA Data Objects for both Extract and Import**
  - Campaign
  - Campaign Definition
  - Communication
  - Communication Definition
  - **Export Definition (New in MA4.4!)**
  - Diagram
  
- **Non file based access**
  - Allows XML Messaging via SAS Event Broker
  - Provides for various message transports
  - Support for Uniform Resource Indicators (URI)
  
- **Enhanced Java API**
  - Establish multiple connections via API Factory
  - Perform multi-threaded operations
  - Support for XML-XSLT processing (See below)
  - Logging Control
  
- **XML Style-Sheet Language (XSLT) Processing**
  - Allows transformation of incoming and outgoing XML to provide for user defined XML schemas
  - Provides Business Level Integration of 3<sup>rd</sup> Party Systems
  - Can be called from Java API (Both 'ad-hoc' and as part of XML submits)
  - Gives easier integration with tools like SAS XML Maps and Microsoft® Excel
  
- **Support for MA Business Contexts**
  - Business Contexts were a new feature of the MA4.3 solution
  - These are supported through the command line and Java API
  - **Support for importing definitions to a particular BC (New in MA4.4!)**
  
- **Support for Campaign Metadata Promotion**
  - Allows you to easily move campaigns between development, test and production OMR repositories
  - Provided via new 'MAPromote' XML Schema
  - Supported through the command line interface via new 'sasmapromote' launcher.
  
- **Improved Error Handling and Logging**

## Section 2. Deployment Overview

The MA integration utilities are a set of client side Java programs that are located outside of the MA Application Server (MACore). There is a utility for extraction of metadata, a utility for importing of metadata, and a utility for promotion of campaign metadata. These utilities are each surfaced by a command line launcher application (.exe file on Windows and .sh scripts on Unix). In addition there is a Java API which allows for 3<sup>rd</sup> party integration.

The MA integration utilities use XML as the primary transport medium to send and receive data to and from the MA metadata repository. This is achieved by utilization of the MA Core API (SAS Marketing Automation Application Server) to access data objects that are persisted within the SAS Metadata Repository (SMR) and DAV. Importing of data is a “one-way” process. A complete XML file that contains data to be added or updated is parsed and processed through MA Core and is then persisted in the repository. Extraction of data is a “two-way” process. An XML file containing a request to extract data is parsed and a resultant XML file containing the information is returned. Both importing and extraction also have the ability to return error messages to the console. Logging to a file system of any errors and processing information is also provided.

The processing logic for extract, import, and promotion is held within a single client Jar file. This jar file is installed with the utilities: `sas.analytics.crm.auxdataio.jar`. The code within this Jar file acts as a client interface to the MA Application Server. Using a connection to MACore allows the utilities to gain access to the campaign and communication information that is persisted in the metadata repository.

The following diagram gives an architectural overview of how the Integration Utilities fit within the MA solution (Note: for clarity not all of pieces of the MA solution are shown).

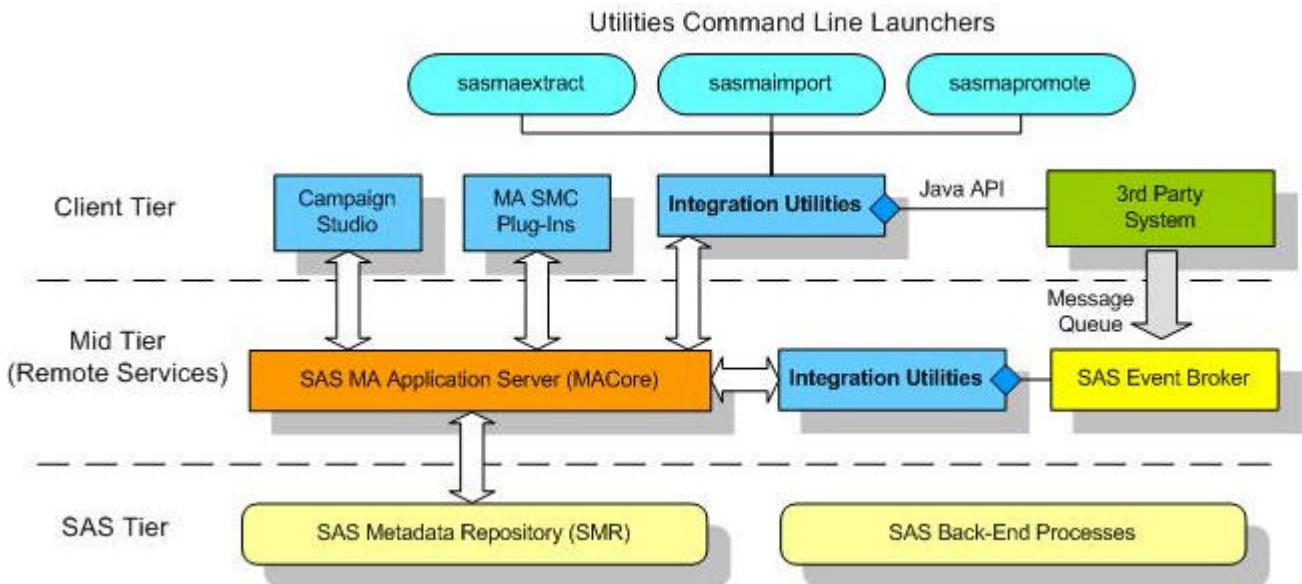


Figure 1 - Architectural Overview.

### Typical Installation of the SAS MA Integration Utilities

It is recommended that Installation of the utilities is performed through the SAS software navigator at the time when the MA environment is initially deployed. The utilities can however be deployed at a later date if required. Although the utilities are essentially a client application (they are found on the MA client 17 CD), they can also be installed to a Mid-tier or SAS-tier server if required. The typical deployed file location for the Utilities is:

`C:\Program Files\SAS\SASMarketingAutomation\MAIntegration\4.1`



In the case of integration with the SAS Event Broker, the utilities are recommended to be installed on the mid-tier since the Event Broker is accessed through SAS Remote Services which normally resides on this tier.

Like other client applications in the MA Solution, the utilities rely on the Java Naming and Directory Interface (JNDI) to establish a connection with the MA Application Server. The section below describes the details of the JNDI property file that is required for this purpose.

## JNDI Properties File

The install process for the utilities creates a populated JNDI (Java Naming & Directory Interface) property file based on the information given through the SAS Software Navigator. This file (normally found in the utilities deployed location) contains Java Naming properties that specify how connection is made to the Marketing Automation Application Server. These properties specify the initial context and provider that will be used to establish the connection. The value of these properties is determined by the type of application server container (BEA WebLogic® or IBM WebSphere®) and the actual machine names and ports on which the server is installed. The file also contains some fixed properties that determine connection and data access timeout values used during communication with the application server.

An example of the JNDI properties file is shown below.

C:\Program Files\SAS\SASMarketingAutomation\MAIntegration\4.1\jndi.properties

```
# Auxdataio jndi.properties

# BEA WebLogic Example:
# java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
# java.naming.provider.url=t3://mahost:7001
#
# IBM WebSphere Example:
# java.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory
# java.naming.provider.url=iiop://mahost:2809

java.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory
java.naming.provider.url=iiop://midserver:2809

# The jnp protocol socket factory class
jnp.socketFactory=org.jnp.interfaces.TimedSocketFactory
# The TimedSocketFactory connection timeout in milliseconds(0 == blocking)
jnp.timeout=0
# The TimedSocketFactory read timeout in milliseconds(0 == blocking)
jnp.sotimeout=0
```

**Figure 4 – Example JNDI Properties File**

In the above example, the application server type is IBM WebSphere® which is running on port 2809 and the server name is 'midserver'. Timeouts for JNP protocol sockets are set to the default value of zero (blocking).

## Manually Deployed Jar Files

Marketing Automation supports mid-tier implementations using either BEA WebLogic® or IBM WebSphere® application server containers. In order to access the services of these containers a number of Jar files need to be manually deployed to the installed location of the Integration Utilities.

The following sections list the required additional Jars for BEA WebLogic® and IBM WebSphere®:



## Additional Jars required for WebLogic®

The installer for the utilities adds these files to the installed location when WebLogic is selected:

- `sas.analytics.crm.ma.core-client.jar`
- `sas.core.jar`
- `wlclient.jar`

## Additional Jars required for WebSphere®

The installer for the utilities adds these files to the installed location when WebSphere is selected:

- `sas.analytics.crm.ma.core-client.jar`
- `sas.core.jar`

The following Jars files need to be copied and moved from the WebSphere Pluggable Application Client to the installed location of the utilities (C:\Program Files\SAS\SASMarketingAutomation\MAIntegration\4.1):

- `bootstrap.jar`
- `ecutils.jar`
- `ffdc.jar`
- `ibmext.jar`
- `ibmorb.jar`
- `idl.jar`
- `iwsorb.jar`
- `iwsorbutil.jar`
- `j2ee.jar`
- `lmproxy.jar`
- `messagingClient.jar`
- `naming.jar`
- `namingclient.jar`
- `ras.jar`
- `sas.jar`
- `utils.jar`
- `wsexception.jar`
- `wssec.jar`

## File I/O permissions

Because the utilities command line interface uses file based XML requests and responses, it is important to ensure that the user session that launches the utilities has both read and write access to the installed location and also the location of any request and/or response files. If read or write permissions are not granted for the user session, file IO exceptions may cause the import/extract process to fail.

## Security Model

The security model used by the utilities is implemented through the underlying mechanisms that are provided to the SAS Application Server by the SAS Metadata Repository. Security permissions are checked on the metadata server and only data for which the user has permissions is returned. The metadata server also provides for an admin user facility, known as an Unrestricted User, which has almost all permissions set. This facility allows access to MA metadata that has been created by all users and would typically be used where import and extract of metadata is performed in batch mode.

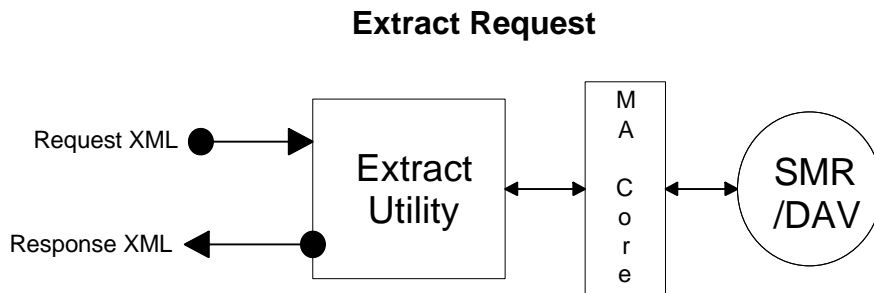
This security model is the same for any client (including SAS Campaign Studio) that uses the SAS Application Server and the SAS metadata repository. No further security has been implemented within the utilities as this was deemed unnecessary for normal operation.

## Intended Usage

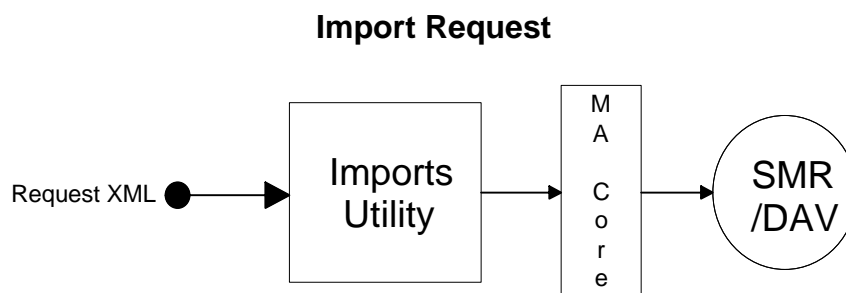
The utilities have been created in response to requirements raised through SAS World-Wide Marketing. These requirements include the integrated mechanisms to import and extract metadata to and from the MA solution for reporting and other 3<sup>rd</sup> party usage. It is anticipated that additional integration effort will be required in order to apply the utilities for a particular use. For example, it will be necessary to convert the XML files to another form (e.g. SAS dataset) so that reporting tools can be applied to the data. SAS XML Mapper is a useful tool for this purpose. Another intended usage of the utilities is for integration with third party systems such as Amdocs® and Aprimo®. This integration will require a 'bridging' mechanism to allow the XML transit between the two systems (Aprimo uses .NET technology). Integration with the SAS Event Broker to 'port' XML messages to and from these types of applications is also supported.

## Process Models

Requests to send and receive metadata are made via XML documents. As described earlier, the process of extracting metadata is two-way; an XML document is passed as input that specifies the required data to extract. The process of importing metadata is one-way; an XML document is passed as input that specifies the new metadata to create. The following diagrams depict these process models.



**Figure 2 – Extract Process.**



**Figure 3 – Import Process.**



## Command Line Interface (CLI)

Primary access to the utilities is via a command line interface (CLI). This interface is supported by the Command Line Launchers that internally use the SAS Private Java Runtime Engine (JRE) to execute the process code held within the utilities Jar file. The CLI for both `sasmaintport`, `sasmaextract` and `sasmapromote` take the following common arguments as input (The required position of each argument on the command line is shown in square brackets []):

[1] SMR user	User ID (including user domain) to log into SMR
[2] SMR password	Password to log into SMR
[3] SMR domain	The SMR domain (e.g., DefaultAuth)
[4] Business Context	The MA Business Context (e.g., Master Business Context)
[5] XML request file	File name and path of the request XML.
[7] Input request style-sheet (Optional)	File name and path of an XSL style-sheet to transform request

In addition the `sasmaextract` and `sasmapromote` utilities can also take an output file argument. This argument must be set for `sasmaextract` but can be omitted for `sasmapromote`. For both utilities, a value of 'none' or an empty pair of quotes ("" ) can also be used to indicate that no output file is specified.

[6] XML output file (Optional)	File name and path of the resultant XML
--------------------------------	---

The `sasmaextract` and `sasmapromote` utilities also support an optional output file style-sheet. Again, a value of 'none' or an empty pair of quotes ("" ) can also be used to indicate that no file is specified.

[8] Output style-sheet (Optional)	File name and path of an XSL style-sheet to transform output
-----------------------------------	--

**Note:** If an optional argument is not required but necessary to keep other arguments in their correct command line position (for example 'output file') then a value of 'none' or empty quotes ("" ) should be used instead.

This CLI can be utilized to launch the applications either interactively or as part as a scheduled ETL batch-job driven from the SAS Scheduling Services and LSF.

## Running the Command Line Interface

To run one of the utilities open up a command prompt from the location where the installation has been deployed and then use a 'dir' command to ensure the correct location. The following files (amongst others) should be visible:

- `jndi.properties` – JNDI property file containing MACore connection details.
- `jndi.properties.orig` – Original (tokenized) version of JNDI property file.
- `sas.analytics.crm.auxdataio.jar` – The utility Jar file.
- `sas.analytics.crm.ma.core-client.jar` – MA Application Server Client Jar
- `sas.core.jar` – SAS Core Jar
- `sas.java.ext.config` – SAS Java extensions Jar
- `sas.launcher.jar` – SAS Launcher Jar
- `sasmaextract.exe` – Extract Launcher
- `sasmaextract.ini` – Extract Launcher initialization file
- `sasmaintport.exe` – Imports Launcher
- `sasmaintport.ini` – Imports Launcher initialization file
- `sasmapromote.exe` – Promotion Launcher
- `sasmapromote.ini` – Promotion Launcher initialization file

The utilities can be executed by providing input arguments to the command line interface. For example, using the extract utility:

```
> sasmaextract mydomain\smrusr smrpwd DefaultAuth "BisCtx" request.xml out.xml
```



Where:

- `mydomain\smrusr` is the SMR user ID (optionally specifying a domain).
- `smrpwd` is the password for the SMR user ID.
- `DefaultAuth` is the SMR domain.
- `BisCtx` is the business context (surround with quotes if contains spaces).
- `request.xml` is the filename of the input XML request.
- `out.xml` is the filename of the output XML response.

**Note:** it is also possible to use batch scripts (.bat files) to execute the utilities. Please see appendix A1 for details.

## XML File Naming Convention

It is good practice to be consistent about the naming convention used for the XML files which will be processed by the utilities. For example, the file names of extract requests could consist of the word “request” and then an underscore followed by the name of the object being extracted. The extracted metadata can be stored to a file that simply consists of the name of that extracted object.

For example, a XML request is created that extracts a campaign called “Camp123\_Dec2005”. The request file would be saved as:

`request_Camp123_Dec2005.xml`

The resultant extracted XML would be saved to a file called:

`Camp123_Dec2005.xml`

This naming convention allows separation between extract requests and extracted data, but also allows the relationship between the two files to be instantly recognized.

## Section 3. XML Schema used by the Integration Utilities

Whilst the XML schema used by the integration utilities is well defined, it is not necessary for the utilities to require an XML Document Template Definition (DTD) file for parsing of the XML. The parser technology within the utilities simply ensures that the XML is “well formed”.

### Document Template Definition (DTD)

The MA Integration Utilities use introspection of the Java API provided by the MA Application Server (MA Core) to determine the structure of metadata objects (DO's) that are stored by MA in the SAS Metadata Repository. The Java code of the MA Application Server is maintained during each formal release of the software. This routine maintenance can result in changes to the interface of this code. The Integration Utilities are able to adapt automatically to these changes by not having a fixed XML schema; the utilities instead allow the schema to “morph” between releases and the current schema is automatically determined by any changes to the interface of the application server. Migration code within the application server also allows for the XML schema from a prior release to be automatically compatible with the XML schema of the next. This provides backward compatibility between releases of the MA software.

This flexibility provides great advantages over a fixed XML schema but can present some issues for parsing of the XML by third party tools – especially for those tools where a DTD is a prerequisite to parse the file. Most (good) XML parsers will read the XML without a DTD. The parser used in the Integration Utilities is from the SAX and DOM Javax libraries and is available in the Sun® JDK. Some tools on the market can “back-engineer” a DTD from the XML file – this is a possible solution for situations where a DTD is necessary.

In absence of a DTD, the following section explains the XML schema used to describe the metadata which comprises a campaign.

### XML Schema for Campaign Objects

When extracted, one or more complete campaigns are encapsulated in the following XML structure:

- |                                 |                                    |
|---------------------------------|------------------------------------|
| • <Batch>                       | Outermost Tag                      |
| o <ResultsSummary>              | Extract Results (Errors etc)       |
| o <ExtractedFrom>               | Source Environment Details         |
| o <LogicalUnit> ...(1 to n)     | Logical Unit of Extracted Metadata |
| ▪ <CampaignDO>                  | Campaign Details                   |
| ▪ <CommunicationDO> ...(0 to n) | Communication Details              |
| ▪ <FlowDO>                      | Diagram Details                    |
| ▪ <typeNodeDO> ...(0 to n)      | Diagram Node Details               |

Within this structure there may exist one or more ‘LogicalUnit’ tags each of which will contain the components that make up the metadata for a single campaign. Each logical unit contains a single ‘CampaignDO’, zero or more CommunicationDO tags and a single ‘FlowDO’ tag. Each logical unit may also contain zero or more ‘NodeDO’ tags. NodeDO’s are represented by particular types of diagram node; for example ‘LinkedNodeDO’ and ‘SelectNodeDO’.

**Note:** The tags outside of the logical unit are explained in later sections of this document. This section of the document focuses just on those components that are held within the ‘CampaignDO’ tag.

## CampaignDO

The CampaignDO tag contains metadata that describes the outline of the campaign. It includes information regarding the campaign name, description, owner, etc, and has nested tags that describe items such as the campaign definition and communication occurrences. It has the following XML structure:

- <CampaignDO>
  - <Name> Campaign name.
  - <Code> Campaign code.
  - <Description> Campaign description.
  - <DateModified> Campaign modification date.
  - <DateCreated> Campaign creation date.
  - <Owner> Valid MA user who owns the campaign.
  - <ModifiedUser> Valid MA user who last modified the campaign.
  - <Shared> Not used. Always true.
  - <HasDiagram> Whether campaign has a flow associated with it. Possible values are: {true, false}
  - <HasCells> Not used. Always false.
  - <HasAudience> Not used. Always false.
  - <HasCommunications> Not used. Always false.
  - <HasCellRefs> Not used. Always false.
  - <Saved> Whether campaign has been saved to SMR. Should always be true
  - <Folder> For contents of this tag see the Folder description.
  - <UserRoles> For contents of this tag see the UserRoles description.
  - <Occurrences> List of occurrences of this campaign. Contains one or more <OccurrenceDO>
  - <Approved> Whether the campaign is approved. Possible values are: {true, false}
  - <LastRunDate> Date last run.
  - <TotalCount> Total count value.
  - <Definition> For contents of this tag see the Definition description.
  - <MinNumOffers> Marketing Optimization (MO) Minimum number of offeres for campaign.
  - <MaxNumOffers> MO Maximum number of offers for campaign.
  - <MinBudget> MO Minimum Budget for campaign.
  - <MaxBudget> MO Maximum Budget for Campaign.
  - <OptimizeStatus> MO Status.
  - <Id> SMR object ID.
  - <VersionNumber> MA version number of this object.
  - <NeedToPersist> Internal mid-tier flag. Possible values are: {true, false}

## Folder

The Folder tag is nested within the CampaignDO tag. It describes the details of the folder that holds the campaign. It has the following XML structure:

- <Folder>
  - <Name> Name of the folder.
  - <DateModified> Modification date of the folder.
  - <ParentFolder> Folder hierarchy above this folder.
  - <Owner> Valid MA user who owns this folder.
  - <VersionNumber> MA version number of this object.



## UserRoles

The UserRoles tag is nested within the CampaignDO tag. It describes the list of users and groups who may access the campaign. The UserRoles tag will contain zero or more UserAndGroupDO tags; one for each user. It has the following XML structure:

- <UserRoles>
  - <UserAndGroupDO> ...(0 to n)
    - <Read> Can read campaign: {true, false}
    - <Edit> Can edit campaign: {true, false}
    - <ViewInPortal> Can view the campaign in Campaign Web Studio: {true, false}
    - <IsAGroup> Is a group rather than a user: {true, false}
    - <Name> SMR User or Group Name (ID).
    - <Id> SMR Object ID.
    - <VersionNumber> MA version number of this object.

## Occurrences

The Occurrences tag is nested within the CampaignDO and CommunicationDO tags. It represents the occurrences of the campaigns or communications that are configured ready for immediate or scheduled execution. The Occurrences tag will contain one or more †OccurrenceDO tags; one for each valid occurrence. It has the following XML structure:

- <Occurrences>
  - <OccurrenceDO> ...(1 to n)
    - <SequenceNumber> Integer occurrence sequence number beginning at 1.
    - <StartDate> Start date of this occurrence.
    - <ExecutionDate> Execution date of this occurrence.
    - <StatusId> Valid status id. See StatusDO description.
    - <Count> Occurrence count.
    - <Executable> Occurrence is executable/exportable: {true, false}
    - <Deployable> Occurrence can be deployed: {true, false}
    - <Executed> Occurrence has ever been executed: {true, false}
    - <Deployed> Occurrence has been deployed: {true, false}
    - <Exported> Occurrence has produced exported files: {true, false}
    - <VersionNumber> MA version number of this object.

## Definition

The Definiton tag is nested within the CampaignDO tag. It represents a copy of the campaign definition that was used when the campaign was originally created. The original contents of the definition would have been set through the SAS Management Console plug-in for creation of campaign definitions. The definition has the following XML structure:

- <Definition>
  - <Name> Campaign definition name.
  - <Description> Campaign definition description.
  - <CheckListItem> List of checklist items.
    - <CheckListItemDO> ...(1 to n) Checklist element. See CheckListItemDO description.
  - <StatusList> List of statuses.
    - <StatusDO> ...(1 to n) Status element. See StatusDO description.
  - <BriefData> Campaign Brief
    - <CodeAutoGenerated> Is Campaign code is automatically generated: {true, false}

† Please Note: The 'Occurence' and 'OccurrenceDO' tags are misspelled in the XML due to a misspelling in the MA Core API's. Please do not attempt to correct these spellings in the XML if the files are to be imported back into the MA system at a later time.



- <Required> Is Campaign code is required: {true, false}
- <Editable> Is Campaign code is editable: {true, false}
- <UserDefinedFields> List of user defined fields.
  - <FieldInfoDO> ...(1 to n) UDF element. See FieldInfoDO description.
- <VersionNumber> MA version number of this brief object.
- <DateModified> Modification date of the definition.
- <ModifiedUser> Valid MA user who last modified this definition.
- <Owner> Valid MA user who owns this definition.
- <ApprovalRequired> If the Approval step is required: {true,false}
- <Schedule> Scheduling element. See 'Schedule' description.
- <Id> SMR object ID.
- <VersionNumber> MA version number of this definition object.
- <NeedToPersist> Internal mid-tier flag.

## CheckListItemDO

The CheckListItemDO is nested within the CheckListItems tag of the campaign definition and represents a single checklist item for the campaign. It has the following XML structure:

- <CheckListItemDO> ...(1 to n)
  - <Position> Position of the checklist item in the definition starting at 1.
  - <Label> Display label for this checklist item.
  - <ItemType> Type: {brief, diagram, sched, approv, execute, report, user}
  - <Description> Checklist item description.
  - <Required> If this checklist item is a required item: {true, false}
  - <UserDefinedFields> List of user defined fields.
    - <FieldInfoDO> ...(0 to n) See FieldInfoDO description.
  - <Status> Status: {notready, ready, incomplete, complete, error}
  - <DateModified> Date the item was last modified.
  - <ModifiedUser> Valid MA user who last modified the item.
  - <VersionNumber> MA version number of this object.

## FieldInfoDO

The FieldInfoDO tag is nested within the UserDefinedFields tag of the check-list items and campaign brief. In both cases it has the following XML structure:

- <FieldInfoDO>
  - <Name> Field name.
  - <DisplayLabel> Display label.
  - <DataType> Field data type. See note [1] below.
  - <NumericType> Numeric type when <DataType> is Numeric. See note [2] below.
  - <PossibleValues> List of possible values where applicable.
  - <DefaultValue> Default value for field.
  - <Value> Actual value entered in the field.
  - <Operator> Used for custom filtering.
  - <Required> If the field is required: {true, false}
  - <Position> Position in the list of fields starting at 0.
  - <FilterField> If the field is used for filtering: {true, false}
  - <VersionNumber> MA version number of this object.



## 1. <DataType>

The field data type has an integer value which corresponds to the following types shown in the list below:  
{0 - Numeric, 1 - Date, 2 - String, 3 - Boolean, 4 - Calculated, 5 - List}

## 2. <NumericType>

The numeric type is used when the data type has a value of zero. Its value corresponds to the following types shown in the list below:

{0 - Integer, 1 - Long, 2 - Float, 3 - Double}

## StatusDO

The StatusDO tag can exist either within the 'StatusList' tag of a campaign definition or within the 'StatusList' tag of a communication definition (under the 'Definition' tag in both cases). For both campaigns and communications these statuses are used in the occurrence information which can be found under the OccurrenceDO tag (see the previous 'Occurrences' description). In either case, the contents of the tag are the same and are shown in the XML structure below:

- <StatusDO>
  - <Type> Campaign (0) or communication (1) status.
  - <Name> Status name. See note [1] below.
  - <Valid> If status is valid: {true, false}
  - <Status> Status index (position in the list). See note [2] below.
  - <Method> If status should be automatically or manually updated: {automatic, manual}
  - <Id> Status Identifier. See note [3] below.
  - <VersionNumber> MA version number of this object.
  - <NeedToPersist> Internal mid-tier flag.

## 1. <Name>

The status name can be one of six pre-defined names or can be user defined. The name of the status represents how the status will appear in the drop-down list of values. The table below shows the pre-defined names that are allowed:

Status Name	Index	Description
Not Ready	1	The Campaign/Communication is not ready for execution
Planned	2	The Campaign/Communication has been planned.
Approved	3	The Campaign/Communication has been approved for execution.
Scheduled	4	The Campaign/Communication has been scheduled ready for execution.
Executed	5	The Campaign/Communication has been executed.
Exported	6	Exports have been created for the Campaign/Communication.
[User Defined]	0	User defined status.

\*Note: Status 5 & 6 occur together and are, therefore, synonymous – a campaign and/or communication are only considered executed when exports have actually been created.

## 2. <Status>

The status tag represents an index to the status code for either campaigns or communications (the index value can appear across both campaigns and communications). The value identifies where the status will appear in the drop-down list. This should not be confused with the status ID (see note [3] below) which represents the unique status code that exists across both campaigns and communications (the ID value must appear only in either campaigns or communications but not both). A value of 'zero' for the status tag indicates that the status is user defined – it will appear towards the end of the list.



### 3. <Id>

The Id of the status represents a unique status code identifier that is spread across campaigns and communications. The status codes for communications must have Id's that do not overlap with those defined for campaigns (and vice-versa). The value in the 'StatusID' of the particular 'OccurrenceDO' maps to the 'Id' of the status code. User defined statuses can have their own unique ID. Typical values for ID are shown below:

```
{0 - Campaign not ready, 1 - Campaign planned, 2 - Campaign approved, 3 - Campaign
scheduled, 4 - Campaign executed, 5 - Communication not ready, 6 - Communication planned,
7 - Communication approved, 8 - Communication scheduled, 9 - Communication executed, 10 -
Campaign exported, 11 - Communication exported.}
```

## Schedule

The schedule tag is nested within the 'Definition' tag and describes the schedule for the campaign. It has the following XML structure:

- <Schedule>
  - <StartDate> Schedule start date.
  - <EndDate> Schedule end date.
  - <ExportDate> Schedule export execution date.
  - <RecurrenceType> Schedule recurrence type: {none, min, hour, day, week, month}
  - <RecurrenceEveryPeriod> Recurrence period value.
  - <RecurrenceWeekDay> Day of the week for weekly recurrence: {mon, tue, ...fri}
  - <RecurrenceMonthDay> Day of the month for monthly recurrence.
  - <RecurrencePeriodType> Recurrence period type: {indef, after, by}
  - <RecurrencePeriodEndAfter> Recurrence period 'end after' value.
  - <RecurrencePeriodEndAfterType> 'End After' list type: {occur, min, hour, day, week, month}
  - <RecurPeriodEndByDate> Recurrence period 'end by' date value.
  - <ExportDataOffsetValue> Value appropriate to the <ExportDataOffsetType>
  - <ExportDataOffsetType> Export offset type: {minutes, hours, days, weeks, months}
  - <CampaignStartBased> Basis for the campaign start date: {startmanual, startcomm}
  - <CampaignEndBased> Basis for the campaign end date: {endmanual, endcomm}
  - <AllowRecurCampaign> If the campaign can recur: {true, false}
  - <ScheduleCommsUsing> Not used.
  - <AllowRecurComms> If the communications can recur: {true, false}
  - <VersionNumber> MA version number of this object.

## NodeDefinitionId Values

The XML structure for CommunicationDOs are not covered in this document but if they are extracted it is useful to know how to find the channel type for the communication. The channel type for a particular CommunicationDO can be determined by the <NodeDefinitionId> field which corresponds to the following fixed set of values. These are all the possible channel types and can be seen in SMC under Marketing Automation\Diagram Tools. The actual text displayed against each one may vary with localization.

15	print ad	22	fax	29	pager mail	36	stmt insert
16	atm	23	general insert	30	phone	37	stmt message
17	calendar	24	call center	31	phone mail	38	tv
18	catalog	25	mail	32	POS branch	39	web
19	electronic	26	mobile phone	33	POS sales force	40	response
20	email	27	multi channel	34	radio		
21	event	28	off the page	35	sales promo		



## Section 4. Using the Extract Utility (sasmaextract)

The command line usage for the extract utility is as follows:

```
sasmaextract <domain\username> <password> <SmrDomain> <contextName> <input file> <output file> [<input stylesheet file> <output stylesheet file>]
```

These arguments must appear in this order. An optional output style-sheet can be specified without setting an input style-sheet by setting the input style-sheet to an empty pair of quotes (""), or by setting it to 'none'. For example:

```
Sasmaextract domain\user pwd DefaultAuth "My Context" request.xml out.xml none outXSL.xsl
```

Before you can run the extract utility, you will first need to create an input request XML file. For example, you may want to create a request that obtains **\*all\*** of the metadata for **\*all\*** of the campaigns currently persisted in the SMR repository. The following simple XML request can be used for this purpose (this should be saved in the 'request.xml' file):

```
<MAExtractRequest>
  <CampaignDO detail="ALL" />
</MAExtractRequest>
```

This request uses the 'detail' attribute of the request to set the level of detail to "ALL". This level brings back all of the detail of the requested metadata including all of the detail of any associated metadata objects. To retrieve campaign metadata information, the 'CampaignDO' object is specified as the target.

**IMPORTANT NOTE ABOUT MEMORY REQUIREMENTS:** The above request may attempt to bring back very large amounts of data when there are many campaigns stored in the metadata repository. This may lead to the process taking up large amounts of memory, causing "Out-of-Memory" and "Corba Marshalling Errors" at the mid-Tier. For this reason, more narrowed searches are recommended. Additional memory (if available) can be allocated to the utilities, if necessary (please see Appendix A2).

### Using Search Criteria (Operator "=")

You can narrow the search criteria by using the 'operator =' attribute to request a particular object by property value. For example, to obtain details of campaigns that are owned by a particular user, create the following request xml file:

```
<MAExtractRequest>
  <CampaignDO detail="ALL">
    <Owner operator="=">John Doe</Owner>
  </CampaignDO>
</MAExtractRequest>
```

This request will bring back all of the campaign objects that belong only to the SMR user called "John Doe". If no campaigns match this criterion, i.e., the user has no campaigns defined, then no data will be brought back but the request will still be successfully completed. If "John Doe" does have at least one campaign defined, this will be returned in the response XML file.

### Using Regular Expressions (Operator "\*\*")

You can also alter the search criteria even further by using the "\*" operator and regular expressions to match particular object properties to a sub-set or range of values. The utilities use regular expression technology provided by the Java 2 Platform, Standard Edition (J2SE), version 1.4. A tutorial on how to form and use regular expressions can be found on the Sun™ web-site at <http://java.sun.com/docs/books/tutorial/extra/regex/index.html>.

For example, to retrieve all campaigns that have a campaign name starting with “Ca”, the following request XML can be used:

```
<MAExtractRequest>
  <CampaignDO detail="ALL">
    <Name operator="*">Ca.+</Name>
  </CampaignDO>
</MAExtractRequest>
```

In the example the regular expression “Ca.+” matches all values of campaign name that start with “Ca” and then continue with any number of different characters.

## Combining Operators and Multiple Properties

It is also possible to use a combination of search operators and properties. For example, if you wanted to search for all campaigns owned by ‘John Doe’ that have campaign names starting with ‘Ca’ the following request XML can be used:

```
<MAExtractRequest>
  <CampaignDO detail="ALL">
    <Owner operator="=">John Doe</Owner>
    <Name operator="*">Ca.+</Name>
  </CampaignDO>
</MAExtractRequest>
```

Here the “=” operator is used to find an exact match for “John Doe” on the “Owner” property of all campaign data objects. This search is further narrowed by use of the “\*” operator to find all of those campaigns from the previous search where the ‘Name’ property matches the regular expression “Ca.+”.

**Note about internal search processing:** the order of the search criteria in the request does not have any effect on the resultant list of data objects or the time taken to process the search. This is because a full list of objects is scanned for matching criteria in one pass. No sub-list processing is performed.

## Obtaining Explicit Detail (detail=“Explicit” attribute)

The ‘EXPLICIT’ detail attribute ensures that only the requested properties of the object and no associated data objects are returned. For example, to return all campaign objects that belong to ‘John Doe’ but only obtain the Owner, and campaign name properties, the following XML can be used:

```
<MAExtractRequest>
  <CampaignDO detail="EXPLICIT">
    <Owner operator="=">John Doe</Owner>
    <Name/>
  </CampaignDO>
</MAExtractRequest>
```

In earlier releases (prior to MA4.3), the ‘ALL’ and ‘EXPLICIT’ attributes could only be used with ‘root-level’ data objects. These are data objects that represent campaigns, campaign definitions, communications, communication definitions, flow (diagram) and node items. Any attempt to change from one level of detail to another was ignored for any nested or associated data objects. This limitation was subsequently removed and these attributes can now be used at all levels.



## Performance versus Detail

The choice between using All or Explicit detail should be balanced against performance required versus the level of detail that is required to be returned from an extract request. Whilst using explicit detail can have performance improvements (since it lessens the amount of XML that needs to be generated by the process) it is also highly restrictive in the level of detail that can be obtained. Various combinations of search type operator, detail attribute and properties can be established that have trade-off's in terms of performance and detail, but these are left as an exercise to the reader.

## Search Criteria Using Folder Details

When searching using folder details it is important to note that the 'operator' attribute must be included in the 'Folder' tag. This ensures that searching will continue at the lower level tags that can exist as properties of a folder. For example to search for all campaigns in a folder named "folder1", the following XML can be used:

```
<MAExtractRequest>
  <CampaignDO>
    <Folder operator="">
      <Name operator="">folder1</Name>
      <ParentFolder operator="">Marketing Automation 4\Data</ParentFolder>
    </Folder>
  </CampaignDO>
</MAExtractRequest>
```

Notice that the folder tag is defined as: `<Folder operator="">` to ensure that searching continues with the lower level tags.

## Searching folders within a particular business context

Where a campaign is held within a particular business context, the folder containing that campaign will reside within a root folder of the chosen business context. This is not immediately obvious from within Campaign Studio because the details of the business context root folder are hidden from the user via the browse dialog. However, the integration utilities do expose this root folder for use in search criteria and within the extracted XML. An example of this is shown the XML below:

```
<MAExtractRequest>
  <CampaignDO>
    <Folder operator="">
      <Name operator="">folder1</Name>
      <ParentFolder operator="">Marketing Automation 4\Business
Contexts\MyBCX\Data\General</ParentFolder>
    </Folder>
  </CampaignDO>
</MAExtractRequest>
```

If the parent folder is specified in an extract request that is aimed at a particular business context, the context specified in this part of the XML must correspond to the business context name that is given on the command line. In the example above, the 'MyBCX' business context is being searched so this must also be specified on the command line. Also notice that the context name in the XML is pre-pended by a folder called 'Business Contexts'.



## Extract Schemas and Searchable Objects

The following XML demonstrates the root-level metadata objects and their associated properties that can be used in search criteria. These examples are available from the ZIP file (CodeSamples.zip) that accompanies this document.

### Campaigns

Requesting a campaign with detail set to "ALL" will result in one or more complete campaigns being extracted. Any, or all, of the search criteria demonstrated below can be used to identify a single or multiple campaigns. A combination of 'operator=' set to "=" or "\*" can also be used to create matching search criteria.

```
<MAExtractRequest>

  <CampaignDO detail="All">
    <Id operator="=">ABCDWXYZABCDWXYZ</Id>
    <Name operator="=">The Campaign Name</Name>
    <Description operator="=">The description of the campaign</Description>
    <Owner operator="=">User Name</Owner>
    <Code operator="=">CAMP_CODE</Code>
    <StartDate operator="=">08/03/05 12:00</StartDate>
    <EndDate operator="=">08/04/05 12:00</EndDate>
    <Folder operator="=">
      <Name operator="=">Folder Name</Name>
      <ParentFolder operator="=">Marketing Automation 4\Data\Parent Folder</ParentFolder>
    </Folder>
  </CampaignDO>

</MAExtractRequest>
```

### Campaign Definitions

Requesting a campaign definition with detail set to "ALL" will result in one or more complete definitions being extracted. Any, or all, of the search criteria demonstrated below can be used to identify single or multiple campaign definitions. As is the case for campaigns, a combination of 'operator=' set to "=" or "\*" can also be used to create matching search criteria.

```
<MAExtractRequest>

  <CampaignDefinitionDO detail="All">
    <Id operator="=">ABCDWXYZABCDWXYZ</Id>
    <Name operator="=">The Campaign Definition Name</Name>
    <Description operator="=">The campaign definition description</Description>
    <Owner operator="=">User Name</Owner>
    <ModifiedUser operator="=">User Name</ModifiedUser>
    <DateModified operator="=">14/06/05 15:42</DateModified>
  </CampaignDefinitionDO>

</MAExtractRequest>
```



## Communications

Requesting a communication with detail set to "ALL" will result in one or more complete communications being extracted. Any, or all, of the search criteria demonstrated below can be used to identify single or multiple communications. As is the case for campaigns, a combination of 'operator=' set to "=" or "\*" can also be used to create matching search criteria.

```
<MAExtractRequest>

  <CommunicationDO detail="All">
    <CampaignId operator="=">ABCDWXYZABCDWXYZ</CampaignId>
    <CampaignName operator="=">The Associated Campaign Name</CampaignName>
    <Id operator="=">ABCDWXYZABCDWXYZ</Id>
    <Name operator="=">The Communication Name</Name>
    <Description operator="=">The description of the communication</Description>
    <Owner operator="=">User Name</Owner>
    <Code operator="=">COMP_CODE</Code>
    <StartDate operator="=">08/03/05 12:00</StartDate>
    <EndDate operator="=">08/03/05 12:00</EndDate>
    <Folder operator="=">
      <Name operator="=">Folder Name</Name>
      <ParentFolder operator="=">Marketing Automation 4\Data\Parent Folder</ParentFolder>
    </Folder>
  </CommunicationDO>

</MAExtractRequest>
```

## Communication Definitions

Requesting a communication definition with detail set to "ALL" will result in one or more complete definitions being extracted. Any, or all, of the search criteria demonstrated below can be used to identify single or multiple communication definitions. As is the case for campaigns, a combination of 'operator=' set to "=" or "\*" can also be used to create matching search criteria.

```
<MAExtractRequest>

  <CommunicationDefinitionDO detail="All">
    <Id operator="=">ABCDWXYZABCDWXYZ</Id>
    <Name operator="=">The Communication Name</Name>
    <Description operator="=">The communication definition description</Description>
    <Owner operator="=">User Name</Owner>
    <ModifiedUser operator="=">User Name</ModifiedUser>
    <DateModified operator="=">14/06/05 15:42</DateModified>
    <Channel operator="=">The Channel Name</Channel>
  </CommunicationDefinitionDO>

</MAExtractRequest>
```



## Export Definitions

Requesting an export definition with detail set to "ALL" will result in one or more complete definitions being extracted. Any, or all, of the search criteria demonstrated below can be used to identify single or multiple communication definitions. As is the case for campaigns, a combination of 'operator=' set to "=" or "\*" can also be used to create matching search criteria.

```
<MAExtractRequest>

  <ExportDefinitionDO detail="All">
    <Id operator="=">ABCDWXYZABCDWXYZ</Id>
    <Name operator="=">The Export Definition Name</Name>
    <Description operator="=">The export definition description</Description>
    <CreatedBy operator="=">User Name</CreatedBy>
    <DateCreated operator="=">14/06/06</DateCreated>
    <ModifiedUser operator="=">User Name</ModifiedUser>
    <DateModified operator="=">15/06/06</DateModified>
    <SubjectID operator="=">Subject_ID_Customer</SubjectID>
    <OutputType operator="=">0</OutputType>
    <InfoMapUrl operator="="> SBIP://Foundation/BIP Tree/ReportStudio/Maps/
      Marketing Automation/theInfoMapName</InfoMapUrl>
  </ExportDefinitionDO>

</MAExtractRequest>
```

## Diagrams

Requesting a diagram (a.k.a. flow data object) with detail set to "ALL" will result in one or more diagrams being extracted complete with nodes that make up the diagrams. Any, or all, of the search criteria demonstrated below can be used to identify a single or multiple diagrams. As is the case for campaigns, a combination of 'operator=' set to "=" or "\*" can also be used to create matching search criteria.

```
<MAExtractRequest>

  <FlowDO detail="All">
    <Id operator="=">ABCDWXYZABCDWXYZ</Id>
    <Name operator="=">The Diagram Name</Name>
    <Owner operator="=">User Name</Owner>
    <DateModified operator="=">14/06/05 15:42</DateModified>
    <DateLastRun operator="=">14/06/05 15:42</DateLastRun>
    <Folder operator="=">
      <Name operator="=">Folder Name</Name>
      <ParentFolder operator="=">Marketing Automation 4\Data\Parent Folder</ParentFolder>
    </Folder>
  </FlowDO>

</MAExtractRequest>
```

## Overriding the Extract Output Location

It is possible to specify the output file of where the extracted XML should be stored from within the extract request. This is simply achieved by specifying the 'output' attribute of the MAExtractRequest tag. The value of the attribute can be set to a file path location or given the value of a Uniform Resource Indicator (URI). This value will override any value which was passed via the command line arguments. The example below shows how the output of the extract process can be diverted to a file located on the local machine.

```
<MAExtractRequest output="C:\SAS\MA\ExtractedCampaigns\MyCampaign.xml">
  <CampaignDO detail="ALL">
    <Name>MyCampaign</Name>
  </CampaignDO>
</MAExtractRequest>
```

**Note:** When extracting single campaigns or other objects, it may make sense (and be good practice) to use a similar output filename as the name of the campaign or object being extracted as in the example above - this however is not mandatory.

## Overriding the XSLT input and output designations

In addition to being able to specify the XML output location, it is also possible to specify the input and output style-sheets (XSL files). This is achieved by setting the 'inxsl' and 'outxsl' attributes. For example, to set the input transformation to a file called 'input.xml' and set the output transformation to 'output.xml' the following request can be used:

```
<MAExtractRequest inxsl="input.xml" outxsl="output.xml">
  <GetCampaign name="MyCampaign" folder="MyFolder"/>
</MAExtractRequest>
```

In this example, the input style-sheet would need to contain code that is able to transform this request to a form that can be properly interpreted by the utilities since the 'GetCampaign' tag is not a native form that is supported. The output style-sheet would contain code that converts the extracted XML to some other form (e.g., an HTML report of the campaign features).

Please see the later section of this document (*Using XML Style-Sheet Transformations*) for more information on using XSLT with the Integration Utilities.

## Extracting Definitions from a particular Business Context

Campaign Definitions, Communication Definitions and Export Definitions can be extracted from the Master Business Context or from another business context in which they have been made available. To extract a definition from a context other than the master, simply specify the chosen context on the command line of the extract command. Note: If the definition is not available from within the chosen context then it will not be extracted. In this case, choose a context where the definition is available or specify the master business context.

## Extracted Metadata Response XML

On successful completion of an extract request, the extracted metadata is written to the response (output) XML file. The format of this file has similar XML structure to the extract request but includes the values of the objects. For example, the following request searches for a campaign identified by a certain name and folder location. The request explicitly requires that the result contains the campaign id, description, owner and code:

```
<MAExtractRequest output="out.xml">
  <CampaignDO detail="EXPLICIT">
    <Name operator="=">Campaign1234</Name>
    <Folder operator="=">
      <Name operator="=">TestCampaigns</Name>
      <ParentFolder operator="=">Marketing Automation 4\Data</ParentFolder>
    </Folder>
    <Id/>
    <Description/>
    <Owner/>
    <Code/>
  </CampaignDO>
</MAExtractRequest>
```

The resultant extracted XML is written to a file called 'out.xml'. The example below shows what this file might look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<Batch>
  <ResultsSummary>
    <ExtractCount>1</ExtractCount>
    <ErrorCount>0</ErrorCount>
  </ResultsSummary>
  <ExtractedFrom>
    <BusinessContext>Master Business Context</BusinessContext>
    <RootDataFolder>Marketing Automation 4\Data</RootDataFolder>
  </ExtractedFrom>
  <CampaignDO>
    <Name>Campaign1234</Name>
    <Code>CAMP1234</Code>
    <Description>One of many test campaigns</Description>
    <Owner>John Doe</Owner>
    <Folder>
      <Name>TestCampaigns</Name>
      <ParentFolder>Marketing Automation 4\Data</ParentFolder>
    </Folder>
    <Id>CDSV0PKIEGKFKAAA</Id>
  </CampaignDO>
</Batch>
```

In reality there could be several other related metadata objects that may be required for extraction. The use of the 'EXPLICIT' detail has limited the resultant data in this example. An important point to note is that the entire response is held within the 'Batch' tags. This allows the same XML to be used by the import process.

## Formatted XML Output

By default, the extract utility produces XML which is formatted with indentation and new-lines so that the structure of the XML is more recognizable and easier to edit in simple text editors. The amount of indentation (number of spaces per indent) is controlled by a property that can be set through a special property file (auxdataio.properties). The formatting can also be turned off by setting the value of this property to zero. See appendix A3 for details on setting of this property.

**Note:** This formatting is not applied if an XSLT style-sheet is used to transform the output XML.

## Logical Units and Object References

Where there is more than one matching metadata object returned by the extract utility, a 'LogicalUnit' tag is used. In the example above, only one matching 'CampaignDO' object is shown as being found by the search so a logical unit is not required. In addition, where multiple objects are returned they may be given an 'object reference'. Object references are zero based and are used to show the inter-relationships that exist which may not be evident from the hierarchy of the XML. Since any metadata object may be referenced more than once, the use of object references also prevent circular dependencies which would result in a 'never-ending' XML document.

## Results Summary

The extracted XML also contains a 'results summary' section which can be used to check how many objects were actually extracted and if any error occurred during processing. In the previous example, the results summary section could be observed at the beginning of the XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Batch>
  <ResultsSummary>
    <ExtractCount>1</ExtractCount>
    <ErrorCount>0</ErrorCount>
  </ResultsSummary>
  . . .
```

In this example, it can be seen that one object was extracted (the single campaign) and that no errors were encountered.

## System Errors Encountered During Processing

In the event that a system error prevents the processing of an extract request to be completed, where possible the utility will attempt to resume processing with the next data object or property. In this situation, the data object or property that could not be extracted will contain a system error message that gives an indication of why the processing failed. These messages are placed within a 'SystemError' tag in the output XML response. The XML takes the form:

```
<SystemError>The reason for the problem is described here</SystemError>
```

These system error tags may appear anywhere within the output XML but will normally be found immediately after the data object or property that caused the problem.

## Business Context and Folder Details

For reference, the extracted XML also contains information about the environment from which the objects have been extracted. The 'ExtractedFrom' section of the XML gives details of the business context and folder from which the object was extracted. This can also be seen in the previous example:

```
. . .
<ExtractedFrom>
  <BusinessContext>Master Business Context</BusinessContext>
  <RootDataFolder>Marketing Automation 4\Data</RootDataFolder>
</ExtractedFrom>
. . .
```



## Section 5. Using the Import Utility (sasmainport)

The command line usage for the import utility is as follows:

```
sasmainport <domain\username> <password> <SmrDomain> <contextName> <input file> [<input  
stylesheet file>]
```

These arguments must appear in this order. An optional input style-sheet can be specified to transform the input XML prior to processing. For example:

```
Sasmainport domain\user pwd DefaultAuth "My Context" request.xml inXSL.xsl
```

### Creating an Import Request XML file

Before you can run the import utility, you will first need to create an import request XML file. The simplest way to create an import request is to use an extracted XML file as a template. The format of the import request is an extension of that given by the extract process.

The import request contains the following elements.

- An optional 'RecoveryLevel' tag which defines how the import behaves when a failure occurs.
- An 'ImportMode' tag which defines how the import will function when an imported object already exists.
- The data to be imported.

### Recovery Level

The recovery level tag is optional. If it is omitted, the default value of 'None' is assumed. This tag can take the following values:

- **None** (default value) If an error is encountered anywhere during import, the import process is abandoned. Objects which have already been imported, however, will remain.
- **LogicalUnit** If an error is encountered during import, the logical unit being imported is abandoned but the utility still attempts to import subsequent logical units.

For example, to set the recovery level to 'LogicalUnit' the following XML is used:

```
<?xml version="1.0" encoding="UTF-8"?>  
<Batch>  
  <RecoveryLevel>LogicalUnit</RecoveryLevel>  
  <LogicalUnit>  
    <CampaignDO>  
      <Name>Campaign with everything stage 2</Name>  
      <Code>Camp1912</Code>  
      . . .
```

## Import Mode

The import mode tag is also optional. If it is omitted, the default value 'New' is assumed. This tag can take the following values:

- **New** (default value) Objects are imported for the first time. If an object of the same type and name, and in the same folder (where appropriate) is found, an error is generated.
- **Delete** Objects are deleted from the system. Where the Delete mode is used within the amend mode, the object identified is deleted from within an existing object. For instance you could delete a named diagram node from within an amended diagram.
- **Replace** The imported object is used to replace any existing object of the same type and folder location (where appropriate). Where a matching object is not already present, it behaves in the same way as New.
- **Amend** This is used to change values in an already existing object. Those values defined within the xml import request are updated into the object(s) named.

For example, to set the import mode to 'Replace' the following XML is used:

```
<?xml version="1.0" encoding="UTF-8"?>
<Batch>
  <ImportMode>Replace</ImportMode>
  <RecoveryLevel>LogicalUnit</RecoveryLevel>
  <LogicalUnit>
    <CampaignDO>
      <Name>Campaign with everything stage 2</Name>
      <Code>Camp1912</Code>
      ...
    </CampaignDO>
  </LogicalUnit>
</Batch>
```

## Example Import Request XML

The sample below shows an example of XML used to import (replace) three Campaign data objects.

```
<?xml version="1.0" encoding="UTF-8"?>
<Batch>
  <ImportMode>Replace</ImportMode>
  <RecoveryLevel>LogicalUnit</RecoveryLevel>
  <ResultsSummary>
    <ExtractCount>3</ExtractCount>
    <ErrorCount>0</ErrorCount>
  </ResultsSummary>
  <ExtractedFrom>
    <BusinessContext>Master Business Context</BusinessContext>
    <RootDataFolder>Marketing Automation 4\Data</RootDataFolder>
  </ExtractedFrom>
  <LogicalUnit>
    <CampaignDO>
      <Name>Campaign with everything stage 2</Name>
      <Code>Camp1912</Code>
      <Description></Description>
      <DateModified>9/1/05</DateModified>
      <DateCreated>9/1/05</DateCreated>
      <Owner>John Doe</Owner>
      <ModifiedUser>John Doe</ModifiedUser>
      <Shared>true</Shared>
      <HasDiagram>true</HasDiagram>
    </CampaignDO>
  </LogicalUnit>
</Batch>
```

```

<HasCells>>false</HasCells>
<HasAudience>>false</HasAudience>
<HasCommunications>>false</HasCommunications>
<HasCellRefs>>false</HasCellRefs>
<Saved>>true</Saved>
<Folder>
  <Name>Verified</Name>
  <ParentFolder>Marketing Automation 4\Data\jbxdoe</ParentFolder>
  <DateModified>5/23/05</DateModified>
  <Owner>John Doe</Owner>
  <VersionNumber>4.4</VersionNumber>
</Folder>
.
.
.
</CampaignDO>
</LogicalUnit>
<LogicalUnit>
  <CampaignDO>
    .
    .
    .
  </CampaignDO>
</LogicalUnit>
<LogicalUnit>
  <CampaignDO>
    .
    .
    .
  </CampaignDO>
</LogicalUnit>
</Batch>

```

For clarity, large amounts of XML have been removed from this sample. In reality there would be several other related metadata objects that would also need to be defined in order for the campaign data object to be imported. An important point to note is that the entire response is held with the 'Batch' tags. This allows multiple data objects to be processed.

## Redundant Data from the Extract Process

The data to be imported should generally take the identical form to the data generated by the extract process. However, some data items are created by the extract process which is redundant for import. The extract process produces additional information that is not required in the import process. *This data can be left in the data to be imported or it can be deleted, but it must not be edited as this will produce unpredictable effects.* The redundant data tags are:

- ResultsSummary - Indicates the number of logical units extracted.
- ExtractedFrom - Indicates the name of the business context and the root data folder.

For details of the 'ResultsSummary' and 'ExtractedFrom' tags please see the previous section that describes usage of the extract utility.

## Importing Definitions to a particular Business Context

New in MA4.4 is the ability to import a business context and make it available in one of the defined business contexts. This is achieved by simply specifying the business context on the import command line. The definition will be imported to the master business context and will also be made available for use in the chosen context. The definition can also be assigned to other contexts by using the MA admin plug-in from with the SAS Management Console.



## Section 6. Using the Campaign Metadata Promotion Utility (*sasmapromote*)

The command line usage for the promote utility is as follows:

```
sasmapromote <domain\username> <password> <SmrDomain> <contextName> <input file> [<output file> <input stylesheet file> <output stylesheet file>]
```

These arguments must appear in this order. The last three arguments are optional but if specified must appear in the correct position. For example, an output style-sheet can be specified without setting an input style-sheet by setting the input style-sheet to an empty pair of quotes (""), or by setting it to 'none':

```
sasmapromote domain\user pwd DefaultAuth "My Context" request.xml out.xml none outXSL.xml
```

### Campaign Metadata Promotion

Campaign Metadata Promotion allows one or more campaigns from one metadata repository to be promoted (copied) to another. When a campaign is promoted, it takes with it all of the other metadata objects it comprises (Diagrams, Communications, Definitions\*, etc). This functionality supports the ability for a Development/Test/Production environment for building campaigns, testing campaigns, and launching campaigns into production.

The promotion utility extracts data from one environment, a 'producer' business context, and loads (imports) it to another environment, a 'consumer' business context in one single step. Each environment is normally represented by separate instances of the SAS Marketing Automation application server and SAS Metadata Repository. Unlike other MA clients, the SAS MA Integration Utilities are equipped with the capability to connect to multiple applications servers within one session. Each application server must however be supported by the same container type (i.e. IBM WebSphere® or BEA WebLogic®). It is not possible to promote campaigns between two environments that are using different kinds of application server container. The SMR domain in both environments must also have the same name (this is normally called 'DefaultAuth').

**WARNING:** Whilst it is possible to promote campaigns between business contexts that are not identical, this is not recommended. To ensure proper functionality of campaigns, promotion should only occur between like-for-like business contexts. Failure to ensure identical business contexts in both environments could result in failure of execution for the promoted campaign(s).

### Creating a Promotion Request

The promotion utility references two business contexts, the producer context and the consumer context.

- **Producer context** - Defined in the command line parameters in the same way as in the extract request.
- **Consumer context** - Defined as part of the promotion request using the 'PromoteTo' tag as explained further on in this section.

The promotion request is an extension of both the extract and import request types. A promotion request contains elements of the extract and import request. This is because campaigns are actually extracted and then re-imported by a promotion request. A promotion request contains the items which are listed below.

- **Search criteria** of the object(s) to be promoted.
- **RecoveryLevel** tag (optional)
- **ImportMode** tag (optional)
- **PromoteTo** identifies the consumer business context to which the data will be promoted. It contains two tags:
  - **NamingProvider** This tag identifies the machine that hosts the data repository.
  - **BusinessContext** This tag gives the name of the consumer business context



'Search criteria' is used and defined in accordance with that specified for an extract request. 'RecoveryLevel' and 'ImportMode' are used in accordance with that specified for an import request. Please see the previous sections of this document for further details of these items.

The 'PromoteTo' section of the request is specific only to promotion requests. This section of the XML describes the business context and environment to which the campaigns identified through the search criteria should be promoted. The business context specified must exist in the specified environment.

## Example Promote Request

The example below shows a request which promotes two campaigns named 'CAMP001' and 'CAMP002'. The search criteria for each campaign are specified within the 'CampaignDO' tags. Each campaign is searched for based on an exact match (operator="=") for the campaign name and stored folder. The detail setting for the request is set to 'ALL' to ensure that all metadata for each campaign is promoted. The recovery level in this request is set to 'LogicalUnit' so that if the promotion of the first campaign fails, the second will be attempted. The import mode is set to 'Replace' so that any existing campaigns in the destination environment with the same name will be overwritten.

```
<?xml version="1.0"?>
<MAPromotionRequest detail="ALL">
  <RecoveryLevel>LogicalUnit</RecoveryLevel>
  <ImportMode>Replace</ImportMode>
  <CampaignDO>
    <Name operator="=">CAMP001</Name>
    <Folder operator="=">
      <Name operator="=">Test</Name>
    </Folder>
  </CampaignDO>
  <CampaignDO>
    <Name operator="=">CAMP002</Name>
    <Folder operator="=">
      <Name operator="=">Test</Name>
    </Folder>
  </CampaignDO>
  <PromoteTo>
    <NamingProvider>t3://mahostprod:7001</NamingProvider>
    <BusinessContext>Master Business Context</BusinessContext>
  </PromoteTo>
</MAPromotionRequest>
```

## Setting the Target Environment and Business Context

The 'PromoteTo' section of the request (highlighted in the example above) is used to specify the target environment (MA Application Server) and business context of where the selected campaigns should be promoted. The content of this section in the request is given below.

### Target Environment

The 'NamingProvider' tag is used to identify the JNDI naming provider which should be used to gain access to the target application server. The information that should be entered between these tags can be obtained from the JNDI property file that is located on the target environment.

**NOTE:** It is **\*not\*** necessary to have the Integration Utilities installed on both the source and target environments but the utilities **\*must\*** be installed and configured for the source environment.



If the Integration Utilities are not installed on the target environment then the same naming provider information can be obtained from a Campaign Studio installation which is configured to access the target environment. Simply open the desired 'jndi.properties' file using a suitable text editor (e.g. Windows® Notepad) and make a note of the naming provider setting. This is identified by the 'java.naming.provider.url' property.

For example, the target environment has clients configured with values in the JNDI property file as follows:

```
...
# IBM WebSphere Example:
# java.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory
# java.naming.provider.url=iiop://mahost:2809

java.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory
java.naming.provider.url=iiop://midserverprod:2809
...
```

The 'NamingProvider' tag in the promotion request would be set as follows:

```
<PromoteTo>
  <NamingProvider>iiop://midserverprod:2809</NamingProvider>
  <BusinessContext>Master Business Context</BusinessContext>
</PromoteTo>
```

## Target Business Context

The 'BusinessContext' tag specifies the business context in the target environment where the campaign(s) should be promoted. This is simply specified as the name of the desired business context. For example, to promote to the same environment as in the example above but to a context called 'Promoted Campaigns' the following would be used:

```
<PromoteTo>
  <NamingProvider>iiop://midserverprod:2809</NamingProvider>
  <BusinessContext>Promoted Campaigns</BusinessContext>
</PromoteTo>
```

## Using XSLT with the Promotion Utility

Input and output XSLT (XML style-sheet) files can be used with the promotion utility. This can be useful for promoting a list of campaigns where the name of the campaigns is stored in another XML format (for example, from a Microsoft® Excel Spreadsheet saved as XML). The output XSLT file could also be used for creating a report (e.g. in HTML) of those campaigns that have been successfully promoted).

See the later section on "*Using XML Style-Sheet Transformations*" with the SAS MA Integration Utilities for further details.



## Section 7. Log Files Created by the Utilities

A log file is created for each run of the utilities that contains the details of any error and/or process messages. This log file can be viewed with a normal text editor (e.g. Notepad on Windows systems). By default, log files for the utilities are created in a 'Log' sub-folder which can be found in the installed location. For example:

```
C:\Program Files\SAS\SASMarketingAutomation\MAIntegration\4.1\Log\
```

A further sub-folder is created based on the user ID that was passed to the command line. This allows individual users quick and simple access to log files that have been created under their own ID's. Each log file is given a unique file name by using the time and date when the process was executed. For example, an admin user with an ID of 'crmadmin' who runs an extract request on 1<sup>st</sup> September 2005 at 9.35am would have a log file created in the following folder...

```
C:\Program Files\SAS\SASMarketingAutomation\MAIntegration\4.1\Log\crmadmin\
```

The log file created would be named:

```
MAExtract_20050901_093504976.log.
```

Where this name is of the form:

```
MAExtract_YYYYMMDD_HHMMSSsss
```

Y = Year  
M = Month  
D = Day  
H = Hours  
M = Minutes  
S = Seconds  
s = Milliseconds

The format of the log file name and the location where log files are created can also be controlled by values provided to a configuration (property) file. This file can also determine the logging level used for the utilities. Please see Appendix A3 for details of setting properties in this file.

**Note:** In the unlikely event that a log file already exists with the same name, the utilities will automatically determine and append a unique digit to the end of the file name in order to separate this from the previous file.

Similar log files are created for import and promote. These file names begin with 'MAImport' and 'MAPromote' respectively. Example content of a partial log file for extract is shown below:

```
Logging to <C:\Program Files\SAS\SASMarketingAutomation\MAIntegration\4.1\log\crmadmin\
MAExtract_20050901_093504976.log>
Starting at Thu Sep 01 09:35:04 BST 2005
SAS MA Extract Utility. Copyright(c) 2004-2005 SAS, Cary, NC, USA. All Rights Reserved.
arg[0] <crmadmin>
arg[2] <DefaultAuth>
arg[3] <>
arg[4] <request_camp001.xml>
arg[5] <camp001.xml>
Parsing file: <request_camp001.xml>
Finished parsing file: <request_camp001.xml>
Start extract request processing
Searched folder: Marketing Automation 4\Data - 20 possible item(s) of the correct type
found
Matching possible item(s) against search criteria...
```



## Section 8. Using XML Style-Sheet Transformations (XSLT)

XML Style-Sheet Language Transformation is essentially a translation mechanism that lets you convert XML data into other forms--for example, into HTML. Different XSL transforms then let you use the same XML data in a variety of ways. The Integration Utilities support XSLT by providing for input and output style-sheet processing either via the command line or via the Java API.

Incoming XML documents can be transformed by the utilities using XSLT to the request formats described in the earlier sections of this document for extract, import and promotion. This allows integration with 3<sup>rd</sup> party systems where the XML generated by those systems is not in the native XML format expected by the utilities. The command line (and Java API) of the utilities support this by providing for an 'input' style-sheet.

XML produced by the extract and promotion utilities can also be transformed to another format such as other XML schemas or HTML. This is also useful for integration with other systems where the information returned by the utilities needs to be in a specific format for further processing. It can also be used to generate HTML reports based on the data extracted or promoted. The command line (and Java API) of the utilities support this by providing for an 'output' style-sheet.

In this section, the use of XSLT will be explained within the SAS MA Integration Utilities by looking at a specific example. This example will show how it is possible to maintain a list of campaigns to be promoted within a Microsoft® Excel spreadsheet, save the spreadsheet as XML and use XSLT to convert this to a well-formed campaign metadata promotion request.

### Programming in XSLT

Before reading the rest of this section the reader is encouraged to understand the basic principles of XSLT and style-sheet programming. This document does not explain the XSLT language itself but does use terms and references that are part of the language specification.

Further information on XSLT can be found on the W3 Consortium web-site at: <http://www.w3.org/Style/XSL/>

### Example: Using XSLT for Promotion Requests

In this example we will develop a style-sheet which is able to transform XML files saved from Microsoft® Excel to the native XML format required by the campaign metadata promotion utility. This style-sheet will allow for a list of campaign names to be held in a spread-sheet so that this can be used as the desired list of campaign to promote. The spreadsheet will also hold information about the target environment and business context that the campaigns should be promoted to.

### Campaign Promotion Spreadsheet

Before we can begin work on the style-sheet we first need to create a spreadsheet to use as the source of the transformation. This spreadsheet must contain all of the information required to perform the promotion. The information held in the spreadsheet must include the following:

#### Campaign Details

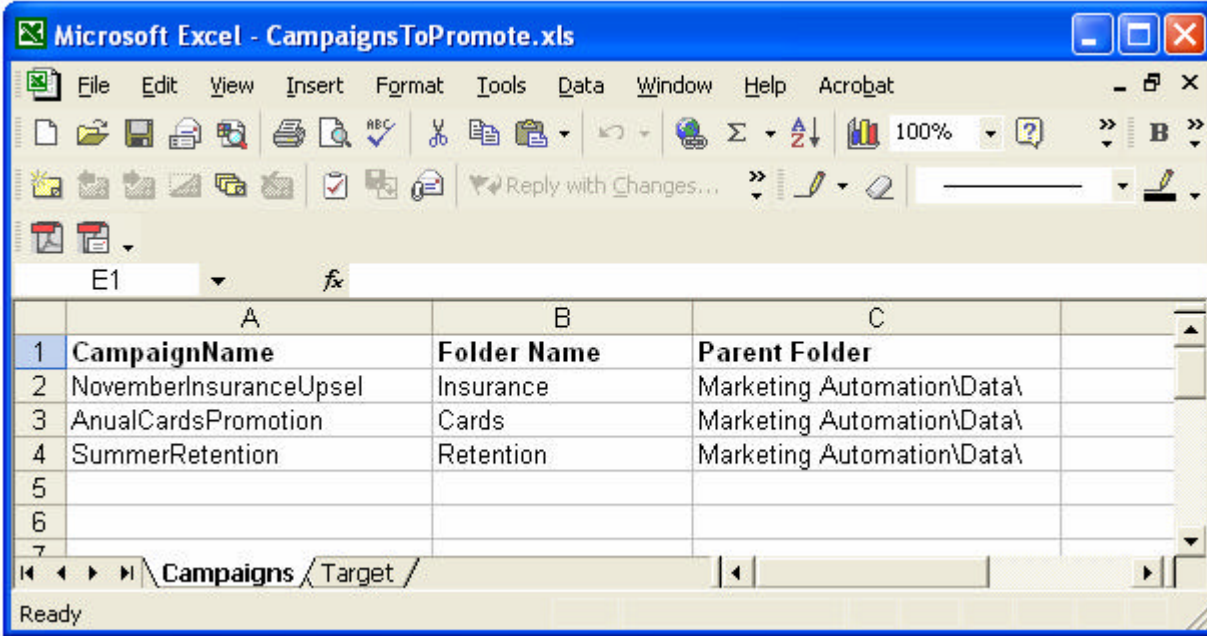
- Campaign Names
- Folder Name (one per campaign name)
- Parent Folder Name (one per campaign name and folder name)

#### Target Details

- Connection (Naming Provider) details of the target environment
- Business Context name to promote the campaigns to

Other information such as the user ID and source business context will be provided through the command line of the promotion utility.

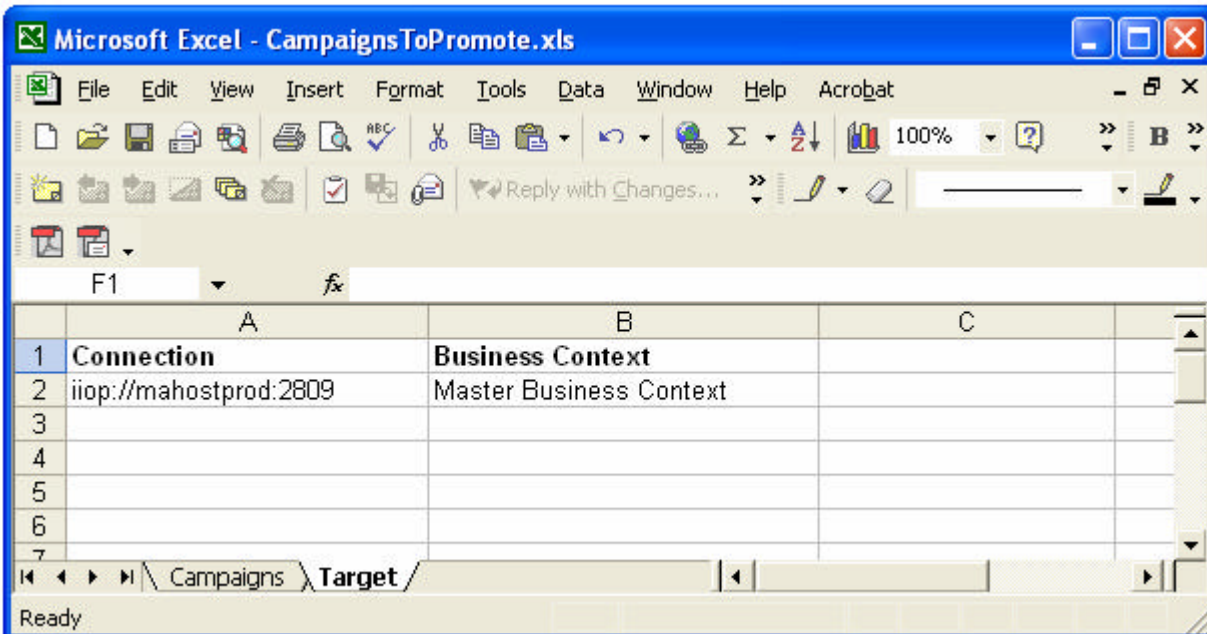
The campaign details and target details can be split across to work-sheets in a single work-book. The campaign details can be held for each campaign on a row that uses three columns. The screen shot below shows an example spreadsheet of this form.



	A	B	C
1	<b>CampaignName</b>	<b>Folder Name</b>	<b>Parent Folder</b>
2	NovemberInsuranceUpsel	Insurance	Marketing Automation\Data\
3	AnualCardsPromotion	Cards	Marketing Automation\Data\
4	SummerRetention	Retention	Marketing Automation\Data\
5			
6			
7			

The 'Campaigns' work-sheet contains information across columns A, B, and C. Each row holds information about a single campaign. In this example there are three campaigns that we want to promote. Notice that the first row of the spreadsheet is simply being used as a column heading; this is something that will need to be handled by the style-sheet since we don't want to create a promotion request for this row.

The following screen shot shows the information in the 'Target' work-sheet:



	A	B	C
1	<b>Connection</b>	<b>Business Context</b>	
2	iiop://mahostprod:2809	Master Business Context	
3			
4			
5			
6			
7			

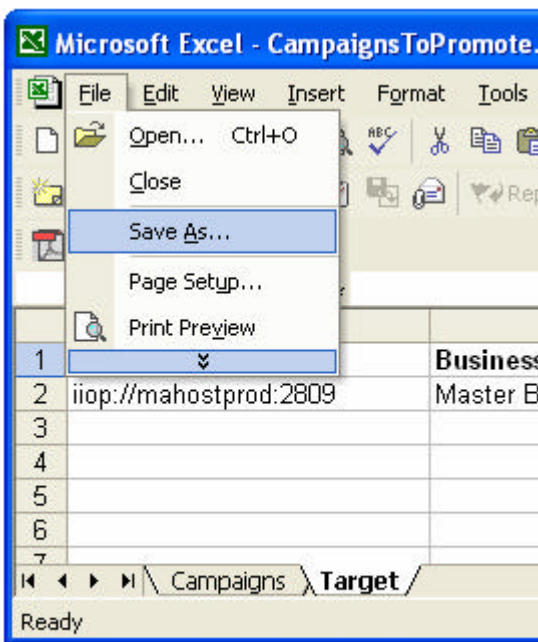
The 'Target' work-sheet simply contains information in columns A and B. The second row of this work-sheet contains the connection naming provider information and business context of the target environment. Again, the first row is being used as a column heading; this will also need to be handled by the style-sheet since we don't want to read information from this row.

## Saving the spreadsheet as XML

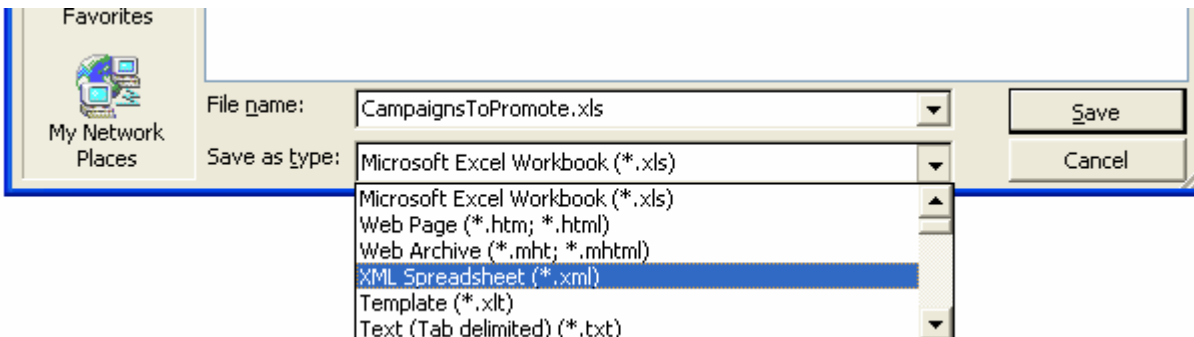
Microsoft® Excel spread-sheets are normally saved in a proprietary Microsoft format (.xls files). Excel does however provide functionality to save and read spreadsheets in other formats. It is possible to save a spreadsheet in XML format by using the 'Save As' feature from the file menu option.

**Note:** Do not confuse the proprietary format of Excel spreadsheets (.xls) with the file extension format used to by XSLT style-sheet files (.xsl). These two file formats contain completely different types of information.

To save the spreadsheet as XML, select File -> Save As:



The save dialog will then appear. Now choose the 'XML Spreadsheet' format from the available file type drop-down list at the bottom of the dialog:



You can optionally change the name of the file if required, but the file extension will be changed automatically to '.xml'. Hit the 'Save' button to save the spreadsheet as XML.



The spreadsheet should now be available in XML format. Since XML files are made of textual characters you can view the contents of this file using a simple text editor such as Windows® Notepad. A complete listing of the XML for this example spreadsheet is given in Appendix B1.

## XML Namespace

The first point to note about the saved spreadsheet XML file is that it uses the namespaces designated by Microsoft Corporation. The declaration of the namespaces is given at the beginning of the XML within the work-book tag:

```
<?xml version="1.0"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:x="urn:schemas-microsoft-com:office:excel"
  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:html="http://www.w3.org/TR/REC-html40">
```

Our style-sheet must be able to process information within the 'excel' (xmlns:x) and 'spreadsheet' (xmlns:ss) namespace. This is because XSLT style-sheets are namespace aware and the XML section we are interested (the rows and columns) are held within the 'Workbook' and 'Worksheet' tags which will be processed within these namespaces:

```
<Worksheet ss:Name="Campaigns">
  <Table ss:ExpandedColumnCount="5" ss:ExpandedRowCount="4" x:FullColumns="1"
    x:FullRows="1">
    <Column ss:AutoFitWidth="0" ss:Width="137.25"/>
    <Column ss:AutoFitWidth="0" ss:Width="97.5"/>
    <Column ss:AutoFitWidth="0" ss:Width="138"/>
    <Column ss:AutoFitWidth="0" ss:Width="104.25"/>
    <Column ss:AutoFitWidth="0" ss:Width="123"/>
    <Row>
      <Cell ss:StyleID="s24"><Data ss:Type="String">CampaignName</Data></Cell>
      <Cell ss:StyleID="s24"><Data ss:Type="String">Folder Name</Data></Cell>
      <Cell ss:StyleID="s24"><Data ss:Type="String">Parent Folder</Data></Cell>
    </Row>
```

## Creating the Transformation Style-Sheet

XSLT Style-sheets are actually just XML files with a pre-determined format. For this reason it is possible to create a style-sheet using a simple text editor. All style-sheets begin with the 'stylesheet' tag and are themselves held within their own 'xsl' namespace. Optionally, we can also specify the output method of the transformation through the 'output' tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml"/>
```

In addition to specifying the 'xsl' namespace, the 'stylesheet' tag must also declare the namespace used in which transformations will take place. In our example, this is in the 'xmlns:x' and 'xmlns:ss' namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:x="urn:schemas-microsoft-com:office:spreadsheet" xmlns:ss="urn:schemas-microsoft-
  com:office:spreadsheet">
  <xsl:output method="xml"/>
```



## Processing the Work-Book Section of the Spreadsheet XML

The first step in the transformation process is to search for and match any 'Workbook' tags in the spreadsheet XML. Finding this first tag will allow the transformation process to recursively process any 'Worksheet' tags that are enclosed within it. The style-sheet script achieves this through addition of the XSL 'template' function as follows:

```
<xsl:template match="x:Workbook">
  <xsl:apply-templates select="x:Worksheet" />
</xsl:template>
```

Notice that the 'Workbook' and 'Worksheet' tags are processed within the 'excel' (x:) namespace. Any 'Workbook' tags that are matched by this template are then further processed by applying other templates that will specifically match the 'Worksheet' tag. Before any work-sheet tags are processed however, we need to start our promotion request output (transformed) XML by including the 'MAPromotionRequest', 'RecoveryLevel' and 'ImportMode' tags. This is achieved simply by inserting these tags within the above template:

```
<xsl:template match="x:Workbook">
  <MAPromotionRequest detail="ALL">
    <RecoveryLevel>LogicalUnit</RecoveryLevel>
    <ImportMode>Replace</ImportMode>
    <xsl:apply-templates select="x:Worksheet" />
  </MAPromotionRequest>
</xsl:template>
```

When the template is processed, any tags that do not form part of the XSL script itself are delivered to the transformation output as they are encountered. Notice that the closing tag of the 'MAPromotionRequest' appears after the request to recursively process any further worksheet templates. This is because this tag needs to be closed in the output XML only after the search criteria and target details have been processed from the worksheet and inserted into the transformed XML.

## Processing the Work-Sheet Section of the Spreadsheet XML

The next step of the transformation process is to recursively process any worksheet information in the spreadsheet XML. This is again achieved by using another XSL template function but this time matching the 'Worksheet' tag:

```
<xsl:template match="x:Worksheet">
  ...Processing of the worksheet goes here...
</xsl:template>
```

The name of the work-sheet is identified by the 'Name' attribute that appears within each 'Worksheet' tag of the spreadsheet XML. The work-sheet section of the spreadsheet XML found within the 'Worksheet' tag is organized into Table, Row and Cell detail. A single 'Table' tag encapsulates multiple 'Row' tags which can in turn contain multiple 'Cell' tags. Each 'Row' tag represents a single row contained within the actual spreadsheet. Each 'Cell' tag contains a 'Data' tag and represents the data of a cell found under a column for each row. The following simplified XML shows this structure:

```
<Worksheet ss:Name="Campaigns">
  <Table>
    <Row>
      <Cell><Data>CampaignName</Data></Cell>
      <Cell><Data>Folder Name</Data></Cell>
      <Cell><Data>Parent Folder</Data></Cell>
    </Row>
    <Row> ...
```



The XSL template that processes the worksheet must be able to identify which work-sheet it is processing. In our example there are just two work-sheets; the 'Campaigns' and 'Target' work-sheets. Using the XSL 'if' function it is possible to determine if the name attribute of the 'Worksheet' tag has a value of 'Target' or 'Campaigns'. This is achieved in our work-sheet template as follows:

```
<xsl:template match="x:Worksheet">

  <xsl:if test="@ss:Name='Target'">
    ...Processing of the Target Work-Sheet goes here...
  </xsl:if>

  <xsl:if test="@ss:Name='Campaigns'">
    ...Processing of the Target Work-Sheet goes here...
  </xsl:if>

</xsl:template>
```

## Processing of the Target Work-Sheet

In order to add the information needed to build the promotion request, each section of the work-sheet template within the XSL 'if' function must now process the rows and columns of the spreadsheet. The XSL 'if' functions provide the ability to process the two work-sheets with differing template code. Using the XPath specification it is possible to apply the template to each row of the work-sheet by using the XSL 'for-each' functionality. We will focus on how this is achieved within the 'Target' section of the work-sheet template:

```
<xsl:template match="x:Worksheet">

  <xsl:if test="@ss:Name='Target'">
    <xsl:for-each select="x:Table/x:Row">
      ...Processing of each row goes here...
    </xsl:for-each>
  </xsl:if>
  ...
</xsl:template>
```

We can further process each cell under the columns of each row by using a second nested 'for-each' block:

```
<xsl:if test="@ss:Name='Target'">
  <xsl:for-each select="x:Table/x:Row">
    <xsl:for-each select="x:Cell/x:Data">
      ...Processing of each row goes here...
    </xsl:for-each>
  </xsl:for-each>
</xsl:if>
...
```

Nesting of these blocks allows transformation processing of the work-sheets in the spreadsheet XML to occur on a row-by-row and cell-by-cell basis. Remember however, that the first row of each work-sheet was being used as a simple header for the columns of the spreadsheet. For the 'Campaigns' work-sheet the first row is used to identify the campaign-name, folder, and parent folder columns. For the 'Target' work-sheet the first row is used identify the connection and business context columns. Since these are actually treated as data by the spreadsheet, they will be included in the spreadsheet XML when the 'SaveAs' feature is used in Excel.



It is possible to conditionally process only certain rows and columns of the spreadsheet XML by using the XSL 'if' function in conjunction with pre-defined XSL expressions and Boolean operators. In our example, we need to determine if the row that is being processed is the first row or a subsequent row. This can be achieved by using the 'position() > 1' expression in the test of an additional XSL 'if' function as follows:

```
<xsl:if test="@ss:Name='Target'">
  <xsl:for-each select="x:Table/x:Row">
    <xsl:if test="position() > 1">
      <xsl:for-each select="x:Cell/x:Data">
        <xsl:if test="position() = 1">
          ...Process the connection cell...
        </xsl:if>
        <xsl:if test="position() = 2">
          ...Process the business context cell...
        </xsl:if>
      </xsl:for-each>
    </PromoteTo>
  </xsl:if>
</xsl:for-each>
</xsl:if>
```

Notice that in addition to checking that the row position is greater than 1, a similar check is made to determine which column (1 or 2) is being processed. This is required for the 'Target' work-sheet to determine if the connection or business context cell is being processed.

The style-sheet code above does not yet produce any output XML for the row and column data. To achieve this for the 'Target' work-sheet we need to add the 'PromoteTo', 'NamingProvider', and 'BusinessContext' tags. In addition we need to output the values of the cells which have been matched from the spreadsheet XML. This is achieved by using the XSL 'value-of' function:

```
<xsl:if test="@ss:Name='Target'">
  <xsl:for-each select="x:Table/x:Row">
    <xsl:if test="position() > 1">
      <PromoteTo>
        <xsl:for-each select="x:Cell/x:Data">
          <xsl:if test="position() = 1">
            <NamingProvider>
              <xsl:value-of select="text()"/>
            </NamingProvider>
          </xsl:if>
          <xsl:if test="position() = 2">
            <BusinessContext>
              <xsl:value-of select="text()"/>
            </BusinessContext>
          </xsl:if>
        </xsl:for-each>
      </PromoteTo>
    </xsl:if>
  </xsl:for-each>
</xsl:if>
```

Notice that the position of the 'PromoteTo' tag in the style-sheet XML is inserted after we match a row but before we match any columns. This ensures that this tag will encapsulate both the 'NamingProvider' and 'BusinessContext' tags in the transformed output XML. The style-sheet code for processing of the 'Target' work-sheet is now complete. The next section describes how we can process the 'Campaigns' work-sheet.



## Processing of the Campaigns Work-Sheet

This section of the style-sheet is quite similar to the previous section that processed the 'Target' work-sheet. The main difference between the 'Target' and 'Campaigns' work-sheet is that there 'n+1' rows required for listing the campaigns. The 'Target' work-sheet simply contains two fixed rows. The complete section of the style-sheet for processing of the 'Campaigns' work-sheet is shown below (it is left as an exercise for the reader to compare this with the style-sheet code from previous section):

```
<xsl:if test="@ss:Name='Campaigns' ">
  <xsl:for-each select="x:Table/x:Row">
    <xsl:if test="position() > 1">
      <CampaignDO>
        <xsl:for-each select="x:Cell/x:Data">
          <xsl:if test="position() = 1">
            <Name operator="=">
              <xsl:value-of select="text()"/>
            </Name>
          </xsl:if>
          <xsl:if test="position() = 2">
            <xsl:text disable-output-escaping="yes">
              &lt;Folder operator="&quot;=&quot;&gt;&lt;/xsl:text>
            <Name operator="=">
              <xsl:value-of select="text()"/>
            </Name>
          </xsl:if>
          <xsl:if test="position() = 3">
            <ParentFolder operator="=">
              <xsl:value-of select="text()"/>
            </ParentFolder>
            <xsl:text disable-output-escaping="yes">&lt;/Folder&gt;&lt;/xsl:text>
          </xsl:if>
        </xsl:for-each>
      </CampaignDO>
    </xsl:if>
  </xsl:for-each>
</xsl:if>
```

The observant reader will notice the use of the XSL 'text' functionality within this portion of the script. This function outputs the 'Folder' tag in the transformed XML. Notice that entity references are used for the '<', '>' and '"' symbols when outputting the tag.

**Note:** Understanding why the XSL 'text' function needs to be used requires knowledge of why XML has to be "Well Formed". It is left as an exercise for the reader to consider why this may not have been the case for the style-sheet XML if the 'Folder' tag had simply been inserted into the style-sheet code. **Clue:** Look at where the 'Folder' closing tag would have been added in relation to other XSL style-sheet tags!

With both work-sheet sections of the style-sheet complete, it simply remains to add the closing tags of the template and style-sheet as follow:

```
</xsl:template>
</xsl:stylesheet>
```

The style-sheet code is now ready to be used with the promotion utility. A complete listing of the style-sheet code can be found in Appendix B2.



## Using the Style-Sheet as input to the Promotion Utility

Recall from the earlier section in this document that the promotion utility command line takes the form:

```
sasmapromote <domain\username> <password> <SmrDomain> <contextName> <input file> [<output file> <input stylesheet file> <output stylesheet file>]
```

In our example we have saved the spreadsheet XML file as 'CampaignsToPromote.XML'. We will assume that the style-sheet code we have developed in this section has been saved to a file called 'promote.xml'. The command line we need to run a promotion request using these files would therefore look like:

```
sasmapromote domain\user pwd DefaultAuth "" CampaignsToPromote.xml none promote.xml
```

Notice that in this example we are not expecting any output XML (extracted campaign metadata) to be generated by the promotion utility. For this reason the 'output file' argument has been set to 'none' to ensure that we maintain the correct position on the command line for the input style-sheet name. Also in this example, notice that we have not set any value for the source business context on the command line, and have instead simply provided an empty set of quotes (""). Using this method, the promotion utility will assume that the campaigns to be promoted are contained in the 'Master Business Context' of the source repository.

When this request is executed the 'CampaignsToPromote.xml' will be transformed using an in-built XSLT processor within the Integration Utilities (Technical note: this processor is provided by the standard Javax.xml libraries of the Sun® JDK). The output of the transformation process will be determined by the contents of our style-sheet that has been provided in the 'promote.xml' file. Internally, the promotion request XML will be created and executed with the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<MAPromotionRequest xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:x="urn:schemas-microsoft-com:office:spreadsheet" detail="ALL">
  <RecoveryLevel>LogicalUnit</RecoveryLevel>
  <ImportMode>Replace</ImportMode>
  <CampaignDO>
    <Name operator="">November Insurance Upsel</Name>
    <Folder operator="">
      <Name operator="">Insurance</Name>
      <ParentFolder operator="">Marketing Automation 4\Data\</ParentFolder>
    </Folder>
  </CampaignDO>
  <CampaignDO>
    <Name operator="">Annual Cards Promotion</Name>
    <Folder operator="">
      <Name operator="">Cards</Name>
      <ParentFolder operator="">Marketing Automation 4\Data\</ParentFolder>
    </Folder>
  </CampaignDO>
  <CampaignDO>
    <Name operator="">Summer Retention</Name>
    <Folder operator="">
      <Name operator="">Retention</Name>
      <ParentFolder operator="">Marketing Automation 4\Data\</ParentFolder>
    </Folder>
  </CampaignDO>
  <PromoteTo>
    <NamingProvider>iiop://mahostprod:2809</NamingProvider>
    <BusinessContext>Master Business Context</BusinessContext>
  </PromoteTo>
</MAPromotionRequest>
```



## Using XSLT with the Extract and Import Utilities

In this section, we have explored the usage of XSLT with the promotion utility to create a method for storing a list of campaigns to promote within a Microsoft® Excel spreadsheet. This is just one of many uses for XSLT with the SAS MA Integration Utilities.

Like the promotion utility, the extract utility supports style-sheets for both input and output processing. The input style-sheet allows extract requests to be provided in XML formats that are non-native to the extract utility. We could for example, provide a similar style-sheet as that developed in this section to transform a spreadsheet that contains a list of campaigns to extract. The output style-sheet can be used to transform the extracted metadata from the native XML format that the utility generates to another format required for another system. For example, to convert to a form that can be directly read into SAS Datasets using the base SAS XML Libref functionality. Another flexibility of XSLT also allows the extracted metadata to be converted to HTML. This could be useful for example to create instant HTML reports of campaign metadata without using any other tools.

The import utility supports input style-sheet processing but does not support output style-sheet processing since no output XML file is generated by this process. The facility to transform the input of an import request provides the ability to import metadata that is not in the native form expected by the import utility. This is useful for integration with other systems where campaign data is generate for adding or updating to the MA environment. This is particularly useful for integration with 3<sup>rd</sup> party Marketing Resource Management (MRM) systems like those provided by vendors such as Aprimo®.

XSLT provides great flexibility to transform the XML data that is used with the Integration Utilities for multiple uses. The list of applications is great and it is left as an exercise of the reader to consider these further. It is hoped that this section of the document has demonstrated some of the advantages of using XSLT with the SAS MA Integration Utilities.

## Section 9. Using the Public Java API

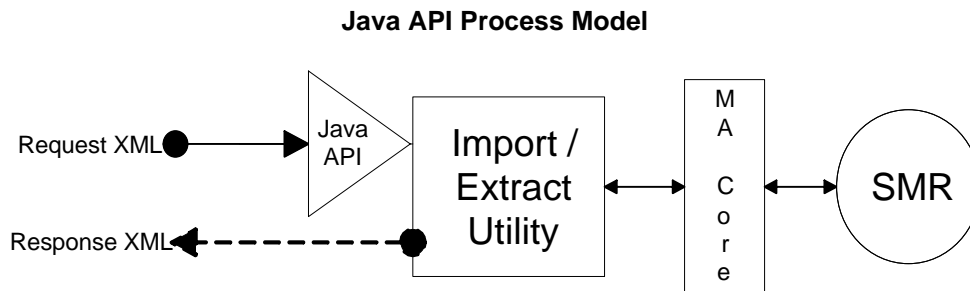
This section assumes that the reader is familiar with Java programming concepts.

The Integration Utilities support a metadata gateway Java API that allows code level integration between the MA solution and other SAS technology or 3rd party systems. Much of the detail given in this section can also be found in the online Java documentation (HTML Javadoc) that accompanies this document. This online documentation can be found in a zip file (emauxdataio\_api43.zip) from the consultant's web site.

The API implements a connection factory pattern that ensures only one connection per combination of SMR user ID, SMR domain, business context and JNDI naming provider is created per application session. This prevents conflicts that may otherwise occur within the Marketing Automation application server. Three API Java classes are provided; the factory class, the connection class and the connection details class. These classes are described in this API documentation. The API allows for a simplified process model to submit import and extract requests in one method call. This process model is explained in more detail below.

### Process Model

The process model for usage of the API is similar to that for the command line interface. It differs in that there is only one request type which can be used to import or extract metadata. The request is parsed by the API and it then automatically determines if the request is for import or extract (Note: currently, the Java API does not support campaign metadata promotion requests). If the request is for extract, the filename of the response is provided within the request. This is defined by the 'output' attribute on the 'MAExtractRequest' tag. The diagram below shows the Java API process model.



### Output Attribute for Extract Requests

The following example XML shows how the output attribute for an extract request can be set:

```
<MAExtractRequest output="C:\MAExtract\Out\CampaignDO_out.xml">
  <CampaignDO detail="ALL"/>
</MAExtractRequest>
```

This request extracts all campaign data objects (and any related data objects) and writes the result to a file called "CampaignDO\_out.xml" in the local folder "C:\MAExtract\Out\".

### API Jar File

No additional Jar files are required in order to support usage of the Java API. The application program interface is exposed by a number of 'public' classes that exist within the utility Jar file (sas.analytics.crm.auxdataio.jar) which is installed as part of the media.



## Using the API to Submit a Request

There are just a few basic steps to submitting a metadata request:

- Create a metadata gateway factory.
- Create a metadata gateway connection details object.
- Using the factory and connection details, build a metadata gateway connection.
- Submit the metadata request.
- Lastly, close the connection when done.

The code below demonstrates this process...

```
import com.sas.analytics.crm.auxdataio.MAMetaGatewayFactory;
import com.sas.analytics.crm.auxdataio.MAConnectionDetails;
import com.sas.analytics.crm.auxdataio.util.MAMetaGatewayConnection;

...

MAMetaGatewayFactory factory = new MAMetaGatewayFactory();
MAConnectionDetails connectionDetails = new MAConnectionDetails(userID, password,
                                                                domain, bisCtx, (String)null);

try {
    MAMetaGatewayConnection con = factory.getConnection(connectionDetails);

    String result = con.submit(inputXML, (String)null, (String)null);
    //If we wanted, we could re-use the connection to make multiple submits here...
    con.close();
}
catch (ApplicationException e) {
    e.printStackTrace();
}

...
```

## Public Classes of the Java API

Most of the public Java API is exposed through three classes:

- MAMetaGatewayFactory
- MAConnectionDetails
- MAMetaGatewayConnection

These classes handle connection to the MA Application Server and submission of metadata requests. In addition to these three main classes, two further supporting classes are provided:

- Log
- MAExtractException

These classes provide control for process logging and allow for trapping of exceptions during the extract process.

The following sections explain each of the above five classes in more detail.



## Metadata Gateway Factory Class

The factory class (`MAMetaGatewayFactory.class`) resides in the root level 'com.sas.analytics.crm.auxdataio.api' package of the API Jar file. The following summary shows the constructor and methods implemented by this class:

### MAMetaGatewayFactory - Constructor Summary

**MAMetaGatewayFactory()**

Construct a metadata gateway connection factory.

### MAMetaGatewayFactory - Method Summary

static void	<b>clearCaches()</b> Clears all of the cached interfaces provided by connection to the MA Application Server.
MAMetaGatewayConnection	<b>getConnection(MAConnectionDetails connectionDetails)</b> Creates a (possibly new) instance of a <code>com.sas.analytics.crm.auxdataio.util.MAMetaGatewayConnection</code> using the connection details provided.
MAMetaGatewayConnection	<b>getConnection(MAConnectionDetails connectionDetails, boolean useExisting)</b> Creates a (possibly new) instance of a <code>com.sas.analytics.crm.auxdataio.util.MAMetaGatewayConnection</code> using the connection details provided.
static java.lang.String	<b>getDefaultContextName()</b> Gets the name of the system default business context.
boolean	<b>hasValidConnection(MAConnectionDetails connectionDetails)</b> Test the connection for validity using the default key name for these connection details.
boolean	<b>hasValidConnection(java.lang.String key)</b> Test the connection for validity.

This class defines a factory API that enables applications to obtain a connection to the MA application server upon which metadata requests can be submitted. This class is NOT guaranteed to be thread-safe, but it does attempt to ensure that only one connection exists per user session. It is up to the user application to make sure about the use of the factory from more than one thread. An application can use the same instance of the factory to obtain one or more instances of the `MAMetaGatewayConnection` provided the instance of the factory isn't being used in more than one thread at a time. If a valid connection already exists based on the connection details provided, then that valid connection instance will be returned by the factory.

To create a metadata gateway factory instance, use the following Java code:

```
MAMetaGatewayFactory factory = new MAMetaGatewayFactory();
```

To use the factory to obtain a metadata gateway connection, use the following Java code.

```
MAMetaGatewayConnection con = factory.getConnection(connectionDetails);
```

Where "connectionDetails", is a populated instance of the `MAConnectionDetails` class.



The factory uses a 'static' hash-map to maintain the list of connections. This ensures that multiple factory classes within an application 'share' the same list. One, and only one, connection object is created for each user ID and domain combination.

## Class Methods

The full method detail of the connection factory class is given below.

### **getConnection(MAConnectionDetails, boolean)**

```
public MAMetaGatewayConnection getConnection(MAConnectionDetails connectionDetails,  
                                             boolean useExisting)  
    throws
```

```
com.sas.analytics.crm.error.client.ApplicationException
```

Creates a (possibly new) instance of a `com.sas.analytics.crm.auxdataio.util.MAMetaGatewayConnection` using the connection details provided.

**Parameters:**

`connectionDetails` - the details of the required connection.

`useExisting` - whether an existing connection with the default key name (user + domain + business context + naming provider) should be used. If an existing default connection is not used, a unique key name will be generated for internal use.

**Returns:**

an instance of the metadata gateway connection.

**Throws:**

`com.sas.analytics.crm.error.client.ApplicationException` - if the connection could not be made.

### **getConnection(MAConnectionDetails)**

```
public MAMetaGatewayConnection getConnection(MAConnectionDetails connectionDetails)  
    throws
```

```
com.sas.analytics.crm.error.client.ApplicationException
```

Creates a (possibly new) instance of a `com.sas.analytics.crm.auxdataio.util.MAMetaGatewayConnection` using the connection details provided.

**Parameters:**

`connectionDetails` - the details of required the connection.

**Returns:**

an instance of the metadata gateway connection.

**Throws:**

`com.sas.analytics.crm.error.client.ApplicationException` - if the connection could not be made.

### **hasValidConnection(String)**

```
public boolean hasValidConnection(java.lang.String key)
```

Test the connection for validity. If fails, but exists and is broken, the connection is removed from the list of available connections.

**Parameters:**

`key` - name the connection is known by.

**Returns:**

true if a valid connect exists for the given key, false otherwise.



## hasValidConnection(MAConnectionDetails)

public boolean **hasValidConnection**(MAConnectionDetails connectionDetails)

Test the connection for validity using the default key name for these connection details. If it fails, but exists and is broken, the connection is removed from the list of available connections.

**Parameters:**

connectionDetails - the connections details to check against.

**Returns:**

true if a valid connect exists for the given connection details, false otherwise.

## clearCaches

public static void **clearCaches**()

Clears all of the cached interfaces provided by connection to the MA Application Server. Client applications would not normally need to call this method of the factory class.

## getDefaultContextName

public static java.lang.String **getDefaultContextName**()

Gets the name of the system default business context. Client applications would not normally need to call this method of the factory class.

**Returns:**

a string containing the name of the system default business context.

## Connection Details Class

The connection details class also resides in the root level 'com.sas.analytics.crm.auxdataio.api' package of the API Jar file. This class defines the connection details required by an instance of the MAMetaGatewayFactory class to obtain a connection to the MA application server. These details are recorded by the factory so that available connections can be reused if the details specified match those for an already instantiated connection. Validation and authentication of these details are performed when a connection is made through the factory and a new MAMetaGatewayConnection is created.

The following summary shows the constructor and methods implemented by this class:

### MAConnectionDetails - Constructor Summary

**MAConnectionDetails**()

Construct a connection detail instance with null values.

**MAConnectionDetails**(java.lang.String user, java.lang.String pass, java.lang.String domain, java.lang.String businessContextName, java.lang.String namingProvider)

Construct a connection details instance with the values given.

### MAConnectionDetails - Method Summary

java.lang.String	<b>getBusinessContextName</b> ()	Gets the business context name.
java.lang.String	<b>getDomain</b> ()	Gets the SMR domain as passed to the constructor.
java.lang.String	<b>getNamingProvider</b> ()	Gets the JNDI naming provider as passed to the constructor.



java.lang.String	<b>getPass()</b> Gets the SMR user password as passed to the constructor.
java.lang.String	<b>getUser()</b> Gets the SMR user name as passed to the constructor.

## Class Methods

The full method detail of the connection details class is given below.

### getUser

```
public java.lang.String getUser()  
    Gets the SMR user name as passed to the constructor.  
Returns:  
    the user name.
```

### getPass

```
public java.lang.String getPass()  
    Gets the SMR user password as passed to the constructor.  
Returns:  
    the SMR user password.
```

### getDomain

```
public java.lang.String getDomain()  
    Gets the SMR domain as passed to the constructor.  
Returns:  
    the SMR domain.
```

### getBusinessContextName

```
public java.lang.String getBusinessContextName()  
    Gets the business context name. This method will return the name of the system default context if the  
    value passed to the constructor was null, an empty string ("") or "none".  
Returns:  
    the business context name.
```

### getNamingProvider

```
public java.lang.String getNamingProvider()  
    Gets the JNDI naming provider as passed to the constructor.  
Returns:  
    the JNDI naming provider.
```

## Metadata gateway Connection class

The connection class (`MAMetaGatewayConnection`) also resides in the root level 'com.sas.analytics.crm.auxdataio.api' package of the API Jar file. This class provides a connection to the MA application server upon which metadata requests can be submitted. Using this class, an application can send and receive MA metadata in the form of XML. An instance of this class can be obtained from the `MAMetaGatewayFactory.getConnection` method. Once an instance of this class is obtained, XML can be submitted from a variety of input sources. These input sources are Strings, Files, and URLs. Note that this class reuses several classes from the javax.xml (DOM & SAX) API. This class is NOT guaranteed to be thread



safe and calls to the submit method are synchronous - a thread will be blocked until this method has returned a result. It is up to the user application to make sure about the use of the connection from more than one thread. An application can use the same instance of the connection to submit one or more requests provided the instance of the connection isn't being used in more than one thread at a time.

**Note:** The class constructor performs actions to connect the SAS Application Server. Whilst it is possible to create instances of this connection class directly, it is recommended that the factory class is instead used to ensure only one connection per user ID and Domain exists at any one time.

To submit a request to extract or import metadata, use the following Java code:

```
con.submit("request.xml");
```

Where 'con' is the connection class object and "request.xml" is the string containing the file name (and path) of the request or is a string containing the XML request itself.

To close the connection and disconnect it from the MA Application Server, use the following Java code:

```
con.close();
```

Where 'con' is the connection class object. Once this method has been executed, the connection can no longer receive requests to import or extract metadata. The connection is removed from the list of available connections that is maintained by the factory class.

The following summary shows the constructor and methods implemented by this class:

<b>MAMetadataConnection - Constructor Summary</b>	
<a href="#">MAMetaGatewayConnection</a> ( <a href="#">MAConnectionDetails</a> connectionDetails)	Construct an instance of a metadata gateway connection.
<a href="#">MAMetaGatewayConnection</a> ( <a href="#">MAConnectionDetails</a> connectionDetails, java.lang.String connectionName)	Construct an instance of a metadata gateway connection.

<b>MAMetadataConnection - Method Summary</b>	
void	<b>close()</b> Closes the connection with the MA application server.
java.lang.String	<b>getKeyName()</b>
com.sas.analytics.crm.security.client.MAUser	<b>getUser()</b> Gets the instance of com.sas.analytics.crm.security.client.MAUser associated with this connection.
void	<b>logOff()</b> Closes the connection with the MA application server.
void	<b>logon()</b> Enables the connection to the MA application server by logging on with the connection details provided to the constructor of this class.

static void	<b>setAppServerType</b> (java.lang.String appServer) Sets the application server type in use by effecting specific VM arguments.
static void	<b>setAppServerType</b> (java.lang.String appServer, boolean overwrite) Sets the application server type in use by optionally effecting specific VM arguments.
java.lang.String	<b>submit</b> (java.lang.String xml) Submit an XML request for processing.
java.lang.String	<b>submit</b> (java.lang.String xml, java.lang.String inXSL, java.lang.String outXSL) Submit an XML request with optional input and output transformation (XSLT) processing.
static java.lang.String	<b>transform</b> (java.lang.String xml, java.lang.String inXSL) Transform some XML from one form to another using XSLT processing.
static java.lang.String	<b>transform</b> (java.lang.String xml, java.lang.String inXSL, java.lang.String outXML) Transform some XML from one form to another using XSLT processing specifying the location to store the result.

## Class Methods

The full method detail of the connection details class is given below.

### Submit(String)

```
public java.lang.String submit(java.lang.String xml)
    throws org.w3c.dom.DOMException,
           javax.xml.parsers.FactoryConfigurationError,
           org.xml.sax.SAXException,
           java.io.IOException,
           com.sas.analytics.crm.error.client.ApplicationException,
           MAExtractException,
           javax.xml.parsers.ParserConfigurationException
```

Submit an XML request for processing. The string passed to this method can be either XML of the request or a valid URI to a file or resource containing the XML request. The content of the XML must be well-formed and will be parsed to determine the nature of the request. Extract and import requests are both supported through this method. All other types of request are rejected. In the case of extract requests, the returned string will contain the results of the processing. This will either be XML or the URI of the resulting XML if an output location was specified in the body of the extract request. In the case of import requests, a string of zero length is returned. An application exception is thrown if the type of request submitted is not supported. Note: this method is synchronous - the calling thread will be blocked until processing of the submitted request is complete and the method has returned a result or empty string.

**Parameters:**

xml - a string representing the xml request itself or the URI of the request.

**Returns:**

the xml results, or the URI of the results, or an empty string.

**Throws:**

`org.w3c.dom.DOMException` - if "exceptional" circumstances are encountered during processing of the XML.

`javax.xml.parsers.FactoryConfigurationError` - if a problem with configuration with the parser factories exists.

`org.xml.sax.SAXException` - if a problem is encountered when parsing the XML - for example, if the XML is not well-formed.

`java.io.IOException` - if an I/O exception of some sort occurs - for example if the output file can not be written.

`com.sas.analytics.crm.error.client.ApplicationException` - if the submitted request is not supported.

[MAExtractException](#) - if an exception occurs during extraction of metadata from the application server.

`javax.xml.parsers.ParserConfigurationException` - if a 'serious' configuration error exists with the XML parser.

**Submit(String, String, String)**

```
public java.lang.String submit(java.lang.String xml,
                               java.lang.String inXSL,
                               java.lang.String outXSL)
    throws org.w3c.dom.DOMException,
           javax.xml.parsers.FactoryConfigurationError,
           org.xml.sax.SAXException,
           java.io.IOException,
           com.sas.analytics.crm.error.client.ApplicationException,
           MAExtractException,
           javax.xml.parsers.ParserConfigurationException
```

Submit an XML request with optional input and output transformation (XSLT) processing. This overloaded version of the `submit` method is identical to that described above with the addition of optional values that contain the information needed to act as source input (XML source or transformation instructions) required to transform the input XML, output XML, or both, to some other form. This functionality is useful where the input XML is not in the desired format ready for submission, or the output XML needs to be converted to a form that is ready for processing by a third-party system.

**Parameters:**

xml - a string representing the xml request itself or the URI of the request.

inXSL - the source containing XSLT instructions for transformation of the input XML - set to `null` if not required.

outXSL - the source containing XSLT instructions for transformation of the output XML - set to `null` if not required.

**Returns:**

the result of the processing, or the URI of the results, or an empty string.

**Throws:**

`org.w3c.dom.DOMException` - if "exceptional" circumstances are encountered during processing of the XML.



`javax.xml.parsers.FactoryConfigurationError` - if a problem with configuration with the parser factories exists.

`org.xml.sax.SAXException` - if a problem is encountered when parsing the XML - for example, if the XML is not well-formed.

`java.io.IOException` - if an I/O exception of some sort occurs - for example if the output file can not be written.

`com.sas.analytics.crm.error.client.ApplicationException` - if the submitted request is not supported.

`MAExtractException` - if an exception occurs during extraction of metadata from the application server.

`javax.xml.parsers.ParserConfigurationException` - if a 'serious' configuration error exists with the XML parser.

### **Transform(String, String)**

```
public static java.lang.String transform(java.lang.String xml,
                                       java.lang.String inXSL)
```

Transform some XML from one form to another using XSLT processing. This statically defined method allows an XML document to be transformed to another form prior to, or after, submission of a request. Since this method is statically defined, it can be called as a class method rather than as an instance method of an active connection. The transformed XML is returned as a string from the call to this method.

#### **Parameters:**

`xml` - the input XML to be transformed.

`inXSL` - the source input (XML source or transformation instructions) required to transform the input XML.

#### **Returns:**

the transformed XML as a string.

### **transform(String, String, String)**

```
public static java.lang.String transform(java.lang.String xml,
                                       java.lang.String inXSL,
                                       java.lang.String outXML)
```

Transform some XML from one form to another using XSLT processing specifying the location to store the result. This overloaded version of the `transform` method is identical to that described above with the addition of an optional output location (URI) of where the transformed result should be stored.

#### **Parameters:**

`xml` - the input XML to be transformed.

`inXSL` - the source input (XML source or transformation instructions) required to transform the input XML.

`outXML` - the output location (URI) of where to store the transformation result - ignored if set to `null`.

#### **Returns:**

the URI of the stored output specified by `outXML` or the transformed XML as a string if `outXML` is `null`.

### **close**

```
public void close()
    throws com.sas.analytics.crm.error.client.ApplicationException
```

Closes the connection with the MA application server. A call to this method renders the instance of this class disconnected from the MA application server. Subsequent calls to the `submit` method will fail if a



call to `close` is successful. The connection can be remade by a call to `logon`. Applications which extend this class may also call the `logOff` method which performs exactly the same functionality as `close`.

**Throws:**

`com.sas.analytics.crm.error.client.ApplicationException` - if the connection could not be closed.

## logon

```
public void logon()
```

```
throws com.sas.analytics.crm.error.client.ApplicationException
```

Enables the connection to the MA application server by logging on with the connection details provided to the constructor of this class.

**Throws:**

`com.sas.analytics.crm.error.client.ApplicationException` - if the connection could not be made.

## logOff

```
public void logOff()
```

```
throws com.sas.analytics.crm.error.client.ApplicationException
```

Closes the connection with the MA application server. A call to this method will render the instance of this class disconnected from the MA application server. Subsequent calls to the `submit` method will fail if a call to `logOff` is successful. The connection can be remade by a call to `logon`. The `close` method performs exactly the same functionality as `logOff`.

**Throws:**

`com.sas.analytics.crm.error.client.ApplicationException` - if the connection could not be closed.

## getUser

```
public com.sas.analytics.crm.security.client.MAUser getUser()
```

Gets the instance of `com.sas.analytics.crm.security.client.MAUser` associated with this connection. This method is provided for convenience - applications should not normally need to use this method.

**Returns:**

the MA user associated with this connection.

## setAppServerType(String)

```
public static void setAppServerType(java.lang.String appServer)
```

Sets the application server type in use by effecting specific VM arguments. This method sets up VM arguments that may be needed for execution against a particular application server (IBM WebSphere or BEA WebLogic). This method should only be called once per VM session and should be performed prior to making any connections with the application server. For this reason, the methods have been statically defined. The input string defines the selected server type and must be one of two values: "bea-weblogic" or "ibm-websphere". The VM arguments are set according to which server type is selected.

**Parameters:**

`appServer` - the app-server type to select ("bea-weblogic" or "ibm-websphere").

## setAppServerType(String, boolean)

```
public static void setAppServerType(java.lang.String appServer,  
                                   boolean overwrite)
```



Sets the application server type in use by optionally effecting specific VM arguments. This overloaded version of this method is identical to the one described above with the addition that the VM arguments can be optionally overwritten if already set to a different value. A value of true for `overwrite` will update the VM argument if it already exists and is currently set to a different value. A value of false will keep the VM argument unchanged if it is currently set to some value. If no value is set the VM argument will be written.

**Parameters:**

`appServer` - the app-server type to select ("bea-weblogic" or "ibm-websphere").  
`overwrite` - overwrite any existing VM argument if true, do not overwrite if false.

### **getKeyName**

```
public java.lang.String getKeyName()
```

**Returns:**

Returns the keyName.

## **Closing Connection Objects**

It is important to note that connection objects should not be left open (when the application terminates) and that they should be closed (using the `.close()` method) at the earliest opportunity. Failure to do this may cause data objects within the metadata server to become locked-out for use by other processes. Closing the connection ensures that any data objects are released.

## **Supporting Classes**

For convenience, two additional support classes are also provided by the Java API. Whilst it is not necessary to instantiate these classes within your own Java code, they are used internally by the utilities and expose some functionality which may be useful in some cases. An outline of these classes is given below. Full details of these classes are provided in the online Java documentation that accompanies the utilities from the consultant's web-site.

### **Logging Class**

This class provides various methods to control how logging is performed during processing of metadata import and extract requests. Methods to control the granularity of logging as well as various methods to write messages to the log file are provided. In addition, methods are also provided which allow console output to be redirected to an active log file.

### **Extract Exception Class**

This class extends the `java.lang.Exception` class to provide a "throwable" exception object for use with extract requests. In the event of an error occurring during extract, an instance of this class will be thrown and can be caught by the encapsulating code. The class is included within the public Java API so that it can be declared within any 'try-catch-finally' portions of application code.



## Appendix A1 – Example batch (.bat) scripts for Windows

The simplest way to run the command line interface of the utilities is by using the launchers (**sasmaextract.exe**, **sasmaintport.exe**, and **sasmapromote.exe**) that are provided in the installed location. These launchers use the SAS Private JRE and an initialization (.ini) file to determine how to run the utilities. This is the recommended and simplest method for running the utilities command line.

Alternatively, command scripts (.bat files on Windows platforms) are also provided (from the consultant's web-site). These script files can be used to tailor execution of the utilities and when deployed against BEA WebLogic® they contain the script code similar to the following:

```
@echo runmaextract version 1.0 for Weblogic (Client Deploy)
@echo off

rem %1 = OMR user
rem %2 = OMR password
rem %3 = OMR domain
rem %4 = Business Context
rem %5 = XML request file
rem %6 = XML output file

set majarpath=.
set jrepath=.
set classpath=

set classpath=%classpath%;sas.analytics.crm.auxdataio.jar
set classpath=%classpath%;%majarpath%\sas.analytics.crm.ma.core-client.jar
set classpath=%classpath%;%majarpath%\sas.core.jar
set classpath=%classpath%;%majarpath%\wlclient.jar

%jrepath%java.exe -classpath %classpath% com.sas.analytics.crm.auxdataio.extract.MAExtract %1 %2 %3 %4
%5 %6

set classpath=

echo.
echo Pausing so the window doesn't go away in case there were errors during execution.
pause
```

**Note:** Similar launcher shell scripts files (.sh) are provided by the installation on UNIX platforms.

## Path setting in Windows Command Scripts for Additional Jars and SAS Private JRE

The utilities use additionally deployed Jar files. The 'majarpath' setting within the command script may need to be changed to reflect the location these additional Jars. The default path is the installed location of the utilities. This should not normally need to be changed but can be if required. To change the path setting, simply edit and re-save the command script with notepad (or a similar editor). In addition it should also be noted that these scripts **do not** use the SAS private JRE by default. It is recommended that the fully qualified path of the private JRE should be included by modification of the 'jrepath' setting. The SAS Private JRE for use with the Utilities is currently Version 1.4.1.

## Increasing Memory

Appendix A2.1 describes how to increase the available memory when using the command line launchers. Increasing memory can also be achieved by modifying the above batch script to include the same VM arguments **-Xmx1024M** and **-Xms1024M**. These should be added to the line starting with 'java.exe':

```
%jrepath%java.exe -Xmx1024M -Xms1024M -classpath %classpath% com.sas.analytics.crm.auxdataio.extract.MAExtract
%1 %2 %3 %4 %5 %6
```

**Note:** You should ensure the machine on which the utilities run has enough memory to cope with this increased usage.



## Appendix A2.1 - Increasing Memory Available to the Utilities

Because the Integration utilities are a client of the MA application server they run in their own copy of the Java Virtual Machine (JVM). In some situations it may become necessary to increase the amount of memory (heap space) that is available to the JVM in order to avoid out-of-memory errors when extracting or importing large amounts of metadata. This can be achieved by modifying the launcher initialization (.ini) file to adjust the VM arguments `-Xmx1024M` and `-Xms1024M`. In MA4.4 these values have been set to a default of 1024M but may be increased as necessary (for example to 2048M).

The command line initialization files can be found in the deployed location and are as follow:

`sasmaextract.ini` – Extract Command Line Initialization File  
`sasmapromote.ini` – Promotion Command Line Initialization File  
`sasmaintport.ini` – Import Command Line Initialization File

The files can be edited with any suitable text editor – for example Windows® Notepad.

**WARNING:** Great care should be taken when modifying these files. Incorrect settings may result in improper functionality of the utilities which could result in loss or corruption of persisted metadata.

The VM arguments can be located on the line starting with 'CommandLineArgs':

```
...  
;-----;  
; Any arguments to pass to the above command ;  
; e.g.(for SAS SMC): ;  
; -Djava.ext.dirs= -cp sas.smc.jar;. com.sas.console.visuals.MainConsole ;  
;-----;  
CommandLineArgs=-Xmx1024M -Xms1024M -Djava.system.class.loader=com.sas.app.AppClassLoader  
-Dsas.app.class.dirs="C:\Program Files\SAS\SASMarketingAutomation\MAIntegration\4.1"  
-Dsas.app.class.path=sas.analytics.crm.auxdataio.jar ...  
...
```

**NOTE:** You should ensure the machine on which the utilities run has enough memory to cope with any increase.



## Appendix A2.2 – Setting the system default locale used by the JVM

Because the Integration utilities are a client of the MA application server they run in their own copy of the Java Virtual Machine (JVM). It is possible to alter the default locale (country) that the JVM uses and thus change the way date formats are handled by the utilities. The JVM will assume the same locale as that of the machine under which it is running but in some circumstances this may not be appropriate – for example date formats in the extracted XML are required in English but the machine hosting the utilities is running in German.

To change the default locale of the JVM, the VM argument '`-Duser.language`' can be used from the launcher initialization (.ini) files which can be found in the deployed location and are as follow:

`sasmaextract.ini` – Extract Command Line Initialization File  
`sasmapromote.ini` – Promotion Command Line Initialization File  
`sasmaintport.ini` – Import Command Line Initialization File

The files can be edited with any suitable text editor – for example Windows® Notepad.

**WARNING:** Great care should be taken when modifying these files. Incorrect settings may result in improper functionality of the utilities which could result in loss or corruption of persisted metadata.

The VM arguments can be located on the line starting with '`CommandLineArgs`'. Simply add the language VM argument with the country code of the locale required.

```
...
;-----;
; Any arguments to pass to the above command ;
; e.g.(for SAS SMC): ;
; -Djava.ext.dirs= -cp sas.smc.jar;. com.sas.console.visuals.MainConsole ;
;-----;
CommandLineArgs=-Duser.language=en -Xmx1024M -Xms1024M
-Djava.system.class.loader=com.sas.app.AppClassLoader
-Dsas.app.class.dirs="C:\Program Files\SAS\SASMarketingAutomation\MAIntegration\4.1"
-Dsas.app.class.path=sas.analytics.crm.auxdataio.jar ...
...
```

For more information on using locales in the Java Platform, please see the Sun web-site:

<http://java.sun.com/developer/technicalArticles/J2SE/locale/index.html>



## Appendix A3 – Optional Configuration File (*auxdataio.properties*)

The 'auxdataio.property' file is a text file as should be created in the working folder for the utilities.

Each of the following properties is set via the syntax: *property-name* = *value*.

The default value (shown below in square brackets) for each property is assumed if the file or property is not provided.

### Logging Properties

`log.level` [LEVEL\_INFO] – Controls the logging level of the command line or sets the default level for the Java API. The complete list of possible values for this property is: LEVEL\_ALL, LEVEL\_CONFIG, LEVEL\_FINE, LEVEL\_FINER, LEVEL\_FINEST, LEVEL\_INFO, LEVEL\_OFF, LEVEL\_SEVERE, and LEVEL\_WARNING.

`log.Extract.logName` [MAExtract] – Sets the filename prefix used to create log files during extract processing.

`log.MAPromote.logName` [MAPromote] – Sets the filename prefix used to create log files during extract processing.

`log.MAImport.logname` [MAImport] – Sets the filename prefix used to create log files during extract processing.

`log.dir` [log] – Sets the folder name of the root folder where log files are stored.

`log.filename.date.format` [yyyyMMddHHmmssSSS] – Sets the format of the date-time stamp in the log file name.

`log.file.extension` [.log] – Sets the file name extension used for log files.

### XML Properties

`XMLFormatter.SpacesPerIndent` [2] – Sets the number of spaces used for indentation in the formatted XML.

`xml.file.encoding` [*system default*] – Sets the named character set mapping between the sequences of sixteen-bit Unicode characters and the sequence of bytes used to create formatted XML files. Valid Character set mappings are given in the table below. If not specified, the system default file encoding is assumed.

The following table is taken from: <http://java.sun.com/j2se/1.4.2/docs/api/java/nio/charset/Charset.html>

Charset	Description
US-ASCII	Seven-bit ASCII, a.k.a. ISO646-US, a.k.a. the Basic Latin block of the Unicode character set
ISO-8859-1	ISO Latin Alphabet No. 1, a.k.a. ISO-LATIN-1
UTF-8	Eight-bit UCS Transformation Format
UTF-16BE	Sixteen-bit UCS Transformation Format, big-endian byte order
UTF-16LE	Sixteen-bit UCS Transformation Format, little-endian byte order
UTF-16	Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark

**Note:** 'UTF-8' is typically used for double-byte character sets (DBCS) found in Asian Pacific locales.

### Process Control Properties

`BusinessContextSystemDefaultName.txt` [Master Business Context] – Defines the name of the system default business context used by the utilities if not explicitly specified in the command line, XML, or Java API.

`extract.rootnode.linkages` [false] – Defines if the root-node and its linkages are extracted as part of campaign diagram metadata. This metadata is not required by the import process so is typically not provided (the default value is set to false). A value of 'true' for this property will result in the root-node and its linkages being extracted for each diagram.

**Warning:** setting this property to true could cause 'out-of-memory' exceptions when extracting large diagrams.



## Appendix B1 – Example Campaign Promotion Spreadsheet XML

```
<?xml version="1.0"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:x="urn:schemas-microsoft-com:office:excel"
  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:html="http://www.w3.org/TR/REC-html40">
  <DocumentProperties xmlns="urn:schemas-microsoft-com:office:office">
    <Author>CM</Author>
    <LastAuthor>CM</LastAuthor>
    <Created>2005-07-01T13:07:52Z</Created>
    <LastSaved>2005-07-07T10:02:19Z</LastSaved>
    <Company>SAS Software Ltd</Company>
    <Version>10.4219</Version>
  </DocumentProperties>
  <OfficeDocumentSettings xmlns="urn:schemas-microsoft-com:office:office">
    <DownloadComponents/>
    <LocationOfComponents HRef="file:///\\\"/>
  </OfficeDocumentSettings>
  <ExcelWorkbook xmlns="urn:schemas-microsoft-com:office:excel">
    <WindowHeight>12270</WindowHeight>
    <WindowWidth>19035</WindowWidth>
    <WindowTopX>0</WindowTopX>
    <WindowTopY>30</WindowTopY>
    <ProtectStructure>False</ProtectStructure>
    <ProtectWindows>False</ProtectWindows>
  </ExcelWorkbook>
  <Styles>
    <Style ss:ID="Default" ss:Name="Normal">
      <Alignment ss:Vertical="Bottom"/>
      <Borders/>
      <Font/>
      <Interior/>
      <NumberFormat/>
      <Protection/>
    </Style>
    <Style ss:ID="s23">
      <NumberFormat ss:Format="@"/>
    </Style>
    <Style ss:ID="s24">
      <Font x:Family="Swiss" ss:Bold="1"/>
    </Style>
  </Styles>
  <Worksheet ss:Name="Campaigns">
    <Table ss:ExpandedColumnCount="5" ss:ExpandedRowCount="4" x:FullColumns="1"
      x:FullRows="1">
      <Column ss:AutoFitWidth="0" ss:Width="137.25"/>
      <Column ss:AutoFitWidth="0" ss:Width="97.5"/>
      <Column ss:AutoFitWidth="0" ss:Width="138"/>
      <Column ss:AutoFitWidth="0" ss:Width="104.25"/>
      <Column ss:AutoFitWidth="0" ss:Width="123"/>
      <Row>
        <Cell ss:StyleID="s24"><Data ss:Type="String">CampaignName</Data></Cell>
        <Cell ss:StyleID="s24"><Data ss:Type="String">Folder Name</Data></Cell>
        <Cell ss:StyleID="s24"><Data ss:Type="String">Parent Folder</Data></Cell>
      </Row>
      <Row>

```

```

<Cell><Data ss:Type="String">November Insurance Upsel</Data></Cell>
<Cell><Data ss:Type="String">Insurance</Data></Cell>
<Cell><Data ss:Type="String">Marketing Automation 4\Data\</Data></Cell>
</Row>
<Row>
<Cell><Data ss:Type="String">Annual Cards Promotion</Data></Cell>
<Cell><Data ss:Type="String">Cards</Data></Cell>
<Cell><Data ss:Type="String">Marketing Automation 4\Data\</Data></Cell>
</Row>
<Row>
<Cell><Data ss:Type="String">Summer Retention</Data></Cell>
<Cell><Data ss:Type="String">Retention</Data></Cell>
<Cell><Data ss:Type="String">Marketing Automation 4\Data\</Data></Cell>
</Row>
</Table>
<WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
<Print>
<ValidPrinterInfo/>
<HorizontalResolution>300</HorizontalResolution>
<VerticalResolution>300</VerticalResolution>
</Print>
<Selected/>
<ProtectObjects>False</ProtectObjects>
<ProtectScenarios>False</ProtectScenarios>
</WorksheetOptions>
</Worksheet>
<Worksheet ss:Name="Target">
<Table ss:ExpandedColumnCount="2" ss:ExpandedRowCount="2" x:FullColumns="1"
x:FullRows="1">
<Column ss:AutoFitWidth="0" ss:Width="111"/>
<Column ss:AutoFitWidth="0" ss:Width="146.25"/>
<Row>
<Cell ss:StyleID="s24"><Data ss:Type="String">Connection</Data></Cell>
<Cell ss:StyleID="s24"><Data ss:Type="String">Business Context</Data></Cell>
</Row>
<Row>
<Cell ss:StyleID="s23"><Data ss:Type="String">iiop://mahostprod:2809</Data></Cell>
<Cell ss:StyleID="s23"><Data ss:Type="String">Master Business Context</Data></Cell>
</Row>
</Table>
<WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
<ProtectObjects>False</ProtectObjects>
<ProtectScenarios>False</ProtectScenarios>
</WorksheetOptions>
</Worksheet>
</Workbook>

```

## Appendix B2 – Campaign Promotion XSLT Style-Sheet

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:x="urn:schemas-microsoft-com:office:spreadsheet"
    xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet">

    <xsl:output method="xml"/>

    <xsl:template match="x:Workbook">
        <MAPromotionRequest detail="ALL">
            <RecoveryLevel>LogicalUnit</RecoveryLevel>
            <ImportMode>Replace</ImportMode>
            <xsl:apply-templates select="x:Worksheet"/>
        </MAPromotionRequest>
    </xsl:template>

    <xsl:template match="x:Worksheet">
        <xsl:if test="@ss:Name='Target'">
            <xsl:for-each select="x:Table/x:Row">
                <xsl:if test="position() > 1">
                    <PromoteTo>
                        <xsl:for-each select="x:Cell/x:Data">
                            <xsl:if test="position() = 1">
                                <NamingProvider>
                                    <xsl:value-of select="text()"/>
                                </NamingProvider>
                            </xsl:if>
                            <xsl:if test="position() = 2">
                                <BusinessContext>
                                    <xsl:value-of select="text()"/>
                                </BusinessContext>
                            </xsl:if>
                        </xsl:for-each>
                    </PromoteTo>
                </xsl:if>
            </xsl:for-each>
        </xsl:if>

        <xsl:if test="@ss:Name='Campaigns'">
            <xsl:for-each select="x:Table/x:Row">
                <xsl:if test="position() > 1">
                    <CampaignDO>
                        <xsl:for-each select="x:Cell/x:Data">
                            <xsl:if test="position() = 1">
                                <Name operator="=">
                                    <xsl:value-of select="text()"/>
                                </Name>
                            </xsl:if>
                            <xsl:if test="position() = 2">
                                <xsl:text disable-output-escaping="yes">
                                    &lt;Folder operator="&quot;=&quot;&gt;</xsl:text>
                                <Name operator="=">
                                    <xsl:value-of select="text()"/>
                                </Name>
                            </xsl:if>
                            <xsl:if test="position() = 3">
                                <ParentFolder operator="=">

```

```
        <xsl:value-of select="text()"/>
    </ParentFolder>
    <xsl:text disable-output-escaping="yes">&lt;/Folder&gt;</xsl:text>
  </xsl:if>
</xsl:for-each>
</CampaignDO>
</xsl:if>
</xsl:for-each>
</xsl:if>
</xsl:template>
</xsl:stylesheet>
```

## **Appendix C – Frequently Asked Questions and Known Issues**

### **Frequently asked Questions (FAQ's)**

- Do Campaign Definitions have to be configured in order to import a campaign?
  - No. When you use Campaign Studio to create a campaign, you select a campaign definition as defined in SMC. This is used as a template for the content of the campaign object. The definition in SMC is never referred to again once the campaign is created. Looking at it from the import point of view the campaign object is constructed using only the information contained in the XML file (including the embedded definition) without any link or reference to any campaign definition object in SMC. As a side note, as well as specifying CampaignDO objects in the import file which will result in campaigns visible in Campaign Studio, you can specify CampaignDefinitionDO objects in the file which will result in campaign definitions visible within SMC and available for use by Campaign Studio.
- Is it possible to import a single communication (outside of a campaign)?
  - Yes, this is possible but is not recommended since any communication imported in this way will not be associated with a particular campaign. This will result in the communication being an 'orphaned' object within the metadata repository. It will not be possible to access the communication through Campaign Studio. It is recommended that communications are always imported or updated within the context of a campaign object.
- What happens if I try to import a campaign code that is already there?
  - This may depend on your particular MA implementation and whether having unique campaign code is a requirement. Each data object (DO) has a unique ID. This is the only field that has a mandatory requirement to be unique when importing data objects. The import process will ignore any ID fields if creating new objects – it will automatically create new values for this field.

### **Known Issues**

The following list of issues is not intended to be definitive but it does identify some of the known issues that may be more commonly encountered with the Integration Utilities. Issues that were identified with the previous releases of the software have been resolved unless they continue to be explicitly listed here. Users should also check relevant SAS notes for other related issues identified in the current release of MA software.

#### **General Issues**

- Byte Order Marks (BOMs) will cause parsing exceptions of XML files.
  - XML request files that are submitted to the integration utilities will not be correctly parsed if the beginning of the file contains hidden Byte Order Marks. An error message stating the "document root element" is missing will be observed. BOM are often added by third party XML generation tools, especially those which are based on the .NET framework. This issue is a known problem of the XML parsing technology that is used in Java 1.4. Please ensure any BOM is removed from the XML file prior to submission to the integration utilities.
- Only UTF8 characters can be used as input to the command line.
  - When providing input to the integration utilities command line, only characters with a UTF8 encoding can be submitted. Characters of other encodings may be misinterpreted by the utilities resulting in incorrect operation of the command. For example, When using the German translated version for MA the Master Business Context is translated to "Master Geschäftskontext". Running an extract job will fail with an error message because of the special character 'ä' which will be natively given in a non UTF8 encoding. A work-around to this problem is to create a .bat script saved in UTF8 format and submit the command by running the script.

## Metadata Extraction Issues

- Large amounts of data may cause abnormal truncation of the extracted XML.
  - Request to extract very large amounts of data (for example, multiple campaign objects) may result in the generated XML being abnormally truncated and thus not well-formed. To avoid this problem, smaller more narrowed requests should be submitted. In addition, it may be useful to further increase the amount of memory space available to the utilities (see Appendix A2.1). In MA4.4 the default amount of memory space has been increased which may help to reduce this problem.
- Extracting from a non-existing folder will create that folder.
  - If an extract request contains a reference to a folder that does not exist, the specified folder will be created when the request is submitted. This problem is limited to the case of specifying a specific, single folder incorrectly and will result in the creation of one folder. The problem only affects sub-folders, NOT root folders or business contexts. The MA system treats the old MA4.3 root folder (Marketing Automation\Data) and the new MA4.4 root folder (Marketing Automation 4\Data) as referring to the new MA4.4 root folder. It doesn't matter which is specified in an extract request or an import request - the correct results will be achieved.
- The XML for an extracted communication definition does not list all of the business contexts with which it is associated.
  - When a communication definition is extracted, only the master business context is listed in the output XML. Even though the communication definition may be referenced in other business contexts, only the master context will be listed inside the <ExtractedFrom> section at the beginning of the XML. The information in this tag is for user information only. It does not have any effect when the definition is re-imported.
- Errors will be encountered when extracting a campaign or diagram that contains a code node using a stored process with a float type as input.
  - This problem occurs for any stored process where a 'float' is specified as input for the code node. The extraction does continue to completion but the XML containing the stored process detail is missing information for the field info data objects. The float value in the list of inputs causes the extract process to abandon the input item and any subsequent input items after the float. The work-around for this is to avoid float type input variables to the stored process - but this is only true if a diagram containing a code node is expected to be manipulated by the integration utilities. This includes the case where such diagrams would be promoted through dev-test-prod. Instead use alternate input types (e.g., string) and provide a conversion to float through the stored process.

## Metadata Import Issues

- Campaigns definitions with an external (table) list item user defined field (UDF) in the campaign brief can be extracted but not properly re-imported (or promoted) by the utilities.
  - If you create a campaign definition in SMC that uses a list type user defined field with a reference to an external list (table) it can be extracted using the integration utilities but can not be successfully re-imported. The definition will be re-imported without error but when the imported version is used to build a campaign the reference to the external list will not work: In the campaign, go to the drop down arrows beside the brief (or other checklist item); then click the paper and pencil icon next to the external list UDF. You will get this error message: "the column, table or library referenced by this field is no longer available".
- The list of definitions given in the MA SMC plug-ins must be refreshed before trying to open a definition that has been recently updated using 'replace' mode.
  - Because replace mode deletes an object and re-imports a new one, the unique internal identifiers that are used by the SMC plug-ins will change. If the list is not refreshed (re-cached) in SMC, the old identifier will be used. This identifier will no longer exist and trying to open the definition in the plug-in will generate an exception. Please ensure the list of definitions in SMC is refreshed before trying to open a definition that has been 'replaced' by the integration utilities.

- An imported campaign shows communications in the schedule window in a different order from those created original campaign.
  - A campaign that is created with at least three communications when extracted and then re-imported will not retain the original order of the communications shown in the schedule window. Note: This behavior is also observed in Campaign Studio when using 'SaveAs...'.
- Import of another user's campaign incorrectly changes ownership of the campaign diagram causing view/edit permissions of the importing user to be incorrectly grayed.
  - When one user imports the campaigns of another, although the original user who created the campaign retains ownership of the campaign they do not retain ownership of the diagram. This prevents the original user from being able to change permissions of the user who imported the campaign. The original user can still delete the campaign. The user who imported the campaign can not delete the campaign. Where possible, users should only import campaigns they own to avoid these issues.

### **Campaign Metadata Promotion Issues**

- It is not possible to promote campaigns from one environment to another where the platform of each environment (e.g., Windows to UNIX) is different.
  - Campaign metadata promotion will normally happen between like-for-like environments. For example, it would not normally make sense to test campaigns running on one platform and have the production environment on another. There is however a use case for moving campaigns where a strategic decision has been taken to change platforms. Attempting to promote between different platforms (even if they are running the same application server container) will results in log-in failures to the destination platform. The work-around is to extract the campaigns and re-import them in two separate steps removing any domain information from user ID as appropriate.
- The initial count columns and optimized count columns of an optimized campaign are not cleared when a campaign is promoted.
  - An optimized campaign (one which has been set up for use with Marketing Optimization) can be successfully promoted without error but it will be noticed that the initial count and optimized counts for the promoted campaign will not have been cleared. Since the promoted campaign is effectively a new optimized campaign the counts should be blank.