Deltek Know more. Do more.™

Deltek Maconomy 2.6.1

Maconomy Tools for Developers

November 25, 2022

While Deltek has attempted to verify that the information in this document is accurate and complete, some typographical or technical errors may exist. The recipient of this document is solely responsible for all decisions relating to or use of the information provided herein.

The information contained in this publication is effective as of the publication date below and is subject to change without notice.

This edition November, 2022.

Contents

Overview

To install universes, reports, and scripts in, and to dump information from, a Maconomy system, certain tools are needed. This manual describes the following tools:

- Maconomy Stamper in “MStamper Reference”
- Maconomy Builder in “MBuilder Reference”
- Maconomy Dumper in “MDumper Reference”

In addition, a short guide to stamping and installing universes and universe reports is provided in “Stamping and Installing Universes and Universe Reports,” along with tips for setting up your editor to quickly stamp and install universes and reports.

This version of the manual describes the functionality of the tools supported by TPU X.

MStamper Reference

Overview

This manual describes how to provide files that are installed on the Maconomy server with a stamp that protects the files from accidental corruption or malicious tampering.

This manual describes MStamper version 3.12.0.

A Maconomy solution can contain a large number of data, format, and script files that are used for reporting, integration, and web-based presentation. The authenticity and integrity of all of these files is paramount to ensuring smooth operation, and information about who wrote what is essential for directing questions and support calls to the right parties.

Maconomy solves these issues by providing all such files with a stamp that protects the files from accidental corruption or malicious tampering. The stamp also contains essential information about the origin of the file, such as author and creation date, and can also be used for restricting the distribution of the file in various ways.

With the release of the MStamper tool, external consultants and Maconomy partner organizations can stamp their products as they see fit, thereby getting the same high level of control of the authenticity and distribution of their work as internal Maconomy developers.

The File Stamp

A file stamp is a block of data that is inserted into the top or near the top of the file that is being stamped. The stamp consists of the following main components.

- **Message digest** — The message digest contains the checksum protection of the file, along with various internal header information. The message digest makes it possible to detect any changes to the file after it was stamped.
- **Tracking information** — Tracking information consists of author and organization fields, creation date and time, and a history field that traces the origin of the license that was used to stamp the file. This information is written in clear text in the stamped file, and can also be accessed through the Maconomy Portal.
- **License information** — License information can be used to restrict the distribution of stamped files as described in “License Model.”

  After a file has been stamped, it should be treated with some caution, especially when archived with certain compression tools or transferred over a network. A stamped file should be treated as binary data, and must therefore not be subjected to new-line or encoding conversions. Some programs will detect stamped files as being text, and this could potentially perform such operations on them. This will subsequently be interpreted as if the file had been altered, and will in effect render it useless.

License Model

The use of stamped files can be controlled in a number of ways. The restrictions stamped into a file are only checked when data from that file interacts with the Maconomy server.

Not all checks are supported by all Maconomy file types. The following table lists the overall contexts in which stamped files can occur and the access checks that are supported on them. See the relevant documentation for each file type for more detailed information.

	MStamper License	MBuilder License	M-Script	Maconomy Server
Add-ons			✓	✓
Customer number		✓	✓	✓
Application version				✓
No license			✓	✓
Login privilege			✓	
Dialog access list			✓	
Machine identification*	✓	✓		
Expiring date	✓	✓		✓
Namespace	✓	✓		

* Currently supported on Windows and Linux only.

Add-Ons

Add-ons in the stamped file are compared against the add-ons that are currently installed on the Maconomy server. A stamped file can contain multiple lists of add-ons, and the server must have all of the add-ons in at least one of these lists.

For example: If a file is stamped with add-ons “64,65” and “92,” the server must have either add-on 64 *and* 65, *or* 92 to use the file.

Customer Number

You can limit the use of a file to a specific set of customers by specifying the customer numbers (also referred to as Server Installation Numbers) for which the file is valid.

Application Version

Certain stamp formats read by the Maconomy server support an optional application version number. If a file is stamped with such a version number, the Maconomy application must have a compatible version.

No License

If a file uses server functionality that is normally restricted by one of the previously described access mechanisms, the “no license” option can be used to mark contents that can be used on all systems, regardless of whether they satisfy these restrictions.

This is a highly privileged option that is not distributed outside Maconomy.

Login Privileges

If a file needs to perform actions as a particular user, you can specify this user in a login privilege. This enables the file to assume the identity of that user on a specific Maconomy installation without having to know the account password.

Dialog Access List

With earlier versions of MStamper, stamped scripts had access to all Maconomy dialogs through the M-Script Maconomy API. As of MStamper 3.2, the Maconomy server dialog access list is checked by default when a stamped script opens a dialog. If the dialog is not in the list, the request is denied, and the script fails.

To preserve backward compatibility with existing M-Script solutions, scripts that are already stamped with an older version of the stamper do not have their dialog access restricted.

You can also disable the dialog access check for newly stamped scripts, but this feature is now controlled by the Tools Access Control system (see Tools Access Control).

Machine Identification

You can restrict the use of a file to a specific host machine, rather than to a Maconomy installation. If the machine has a network adapter with a MAC (Media Access Control) address, this address can be used to identify the machine on which the file can be used.

This check is only supported on Windows and Linux machines.

Expiring Date

An expiration date can be useful for certain time-dependent operations. A file that is stamped with an expiration date becomes invalid at the end of the specified date.

Namespace

To avoid relation name clashes when using the MBuilder tool to install custom relations that are written in the Maconomy Object Language (MOL), and action name clashes when installing Maconomy entity extensions that are written using the Maconomy Entity eXtension Language (MEXL), a four-character namespace is introduced to uniquely identify the object developer organization. The namespace takes the form of three alphanumeric characters and an underscore, for instance `mns_`, and is provided by Maconomy Development. It is written into the license file using the `Namespace` field.

Tools Access Control

The stamping tool relies on the Maconomy Tools Access Control (TAC) system. The general idea of this is that outside the Maconomy development organization a license must be provided for the tool to work. This license can only be obtained through a TAC license file. Such license files can be used to grant the stamping tool certain rights, thus making it possible to distribute the tool to consultants, partners, and customers.

The License File

The TAC license file enables you to stamp files with the stamping tool outside the Maconomy development organization. A license file contains information about who it is licensed to, when it is issued, and the privileges that it grants to the licensee. You can also restrict the validity of the license file by tying it to a given host machine and specifying when it will expire.

By default the stamping tool looks for a license file named `license.tac` in the current working directory. This file can identify the user and usually supplies the stamping tool with the license of this user. If another license file should be used instead, you can specify this to the stamping tool directly on the command line with the `-L / --license` option (see “Command-Line Options Overview”).

The Stamped File

When a file is being stamped, it receives a subset of the TAC fields and access rights read from a license file. As a general rule, access rights must always be passed explicitly from the license to the file that is being stamped, and the stamped file can never get more rights than those present in the license file. Some TAC fields are transferred implicitly to the stamped file but can be explicitly overridden, and a few fields are auto-generated and cannot be modified.

Because access rights are not “inherited” from the license file, it is essential that the properties of the TAC fields described in “TAC Field Overview” are understood, to pick out the fields and values that should be included in the stamped file.

Compatibility Issues

The set of TAC fields is not static. Additions and changes can be expected on a regular basis. These changes can generally be expected to be backward-compatible in the sense that files that are stamped with previous versions of the stamping tool continue to work with newer versions of the Maconomy technology suite.

However, the opposite cannot be guaranteed: files that are stamped with a newer version of the stamping tool might be rejected by older applications, because the stamp contains fields that the older application does not recognize. If possible, files should therefore always be stamped with the stamping tool from the same TPU version as the installation on which the files are to be deployed.

However, it may not be possible to use the old stamping tool if the license file that is being used has undergone changes that prevent the old stamping tool from reading it. In such cases, a newer stamping tool must be used, and if the target Maconomy installation cannot read the stamped files that are produced by this newer stamping tool, a TPU upgrade of the Maconomy installation is the only solution.

To reduce the frequency of these incompatibilities, two kinds of TAC fields exist:

- Mandatory TAC fields must be recognized and understood by the stamping tool and all applications that read stamped files. If a mandatory TAC field is not recognized, the file is rejected. These fields can be recognized by the `TAC_` prefix.
- Optional TAC fields can be ignored if the stamping tool or application that is reading the stamped files does not recognize them. If the application does recognize the TAC fields, it must respect any restrictions that are imposed by these fields. These fields can be recognized by the `TAC*` prefix.

In this way, the introduction of a new optional TAC field will not break backward compatibility with earlier applications, whereas the introduction of a mandatory TAC field will.

TAC Field Overview

Some TAC fields can be specified on the command line with the option `-T / --TAC` (see “Command-Line Options Overview”.) Others are auto-generated by the stamping tool and only appear in the license file or stamped output. The following tables list all of these fields, along with their purposes and whether their values can be edited. The names of TAC fields are case-sensitive and must appear exactly as shown in the following table.

Some TAC fields can occur in both license files and ordinary stamped files; others are reserved for the license files. The following table shows the fields that can occur both places, while the next table shows the license file-only fields. In both tables, the Edit column indicates whether you can edit the field's value.

Field ID	Edit	Description
AddOns	Yes[2]	A comma-separated list of add-on numbers that are required on the server. Several such lists can be specified.
AllowIgnoreDialog-AccessControl	Yes[3]	A Boolean value that specifyies whether the script is allowed to disable dialog access control when opening dialogs. This enables the script to access dialogs that the user is otherwise not allowed to open.
ApplicationVersion	Yes	The Maconomy application version or range of versions for which this file is valid. Syntax: `<Lang> [ <MajorVer> [ . <MinorVer> ] ].` This field is only supported for certain server-side file types such as DDL and dialog access lists.
Author	Yes[1]	The identity of the licensee.
CreationDate	No	The current date (D.M.Y).
CreationTime	No	The current time (H:M:S).
CustomerNumber	Yes[2]	A customer number for which the licensee is allowed to stamp scripts. More than one entry may exist. (See command-line option: `-c / --customernumber` in "Command-Line Options Overview".)
Format	Yes[2]	Identifies the format of the stamped file. Applications that read the stamped file might impose restrictions on the file formats that they accept.
History	No	Creation history of the stamped file. More than one entry may exist. These fields come before any other entries, and the top-most entry is the most recent.
IsLicense	No	This field is used to prevent a stamped file from being misused as a license file, because the formats might otherwise be identical.
LoginPrivilege	Yes[2]	Special login privileges on the Maconomy server. M-Script only; see command-line option -l / --login in "Command-Line Options Overview."

Field ID	Edit	Description
MaconomyHQ	Yes[3]	A Boolean value that specifies whether the stamped file was created by the Maconomy development organization. All of the stamped files that are released by Maconomy based on TPU 50 or later have this field set to `yes`. If it is omitted on the command line, it defaults to `no`, but if it is absent in a stamp, it defaults to `yes`.
Namespace	Yes[3]	An entity and relation namespace in which the licensee is allowed to add customer relations or extended entities to a Maconomy installation with the MBuilder tool.
NoLicenseRequired	Yes[3]	A Boolean value that specifies whether license requirements on the server apply to the stamped file. If this field is set to `yes`, the stamped file is allowed to circumvent all license checks on the server. If absent, it defaults to `no`.
Organization	Yes[1]	The organization of the licensee.
UseDialogAccessList	Yes[3]	A Boolean value that specifies whether the Maconomy server's dialog access list should be checked when scripts invoke the MScript Maconomy API. If this field is set to `yes`, the dialogs that are accessed by the script must be in the dialog access list. If it is omitted on the command line, it defaults to `yes`, but if it is absent in a stamp, it defaults to `no`.
Version	No	The version number can be used to maintain backward compatibility with TAC information that was written by earlier versions of the stamping tool. It can also be used to deprecate an old stamping tool, forcing the user to upgrade. The stamper will refuse to process a file that has a version number that is higher than its own.

[1] Value can only be changed when creating a license file.

[2] Value must be in the set that is specified in the license file.

[3] Value in stamped file is restricted by the value in the license file.

Field ID	Edit	Description
ExpiringDate	Yes[2]	The date on which the license file expires. The default is the current date. The format is D.M.Y, with the empty string being equal to "never."
FormatPrivileges	Yes[1]	Specific format privileges that are granted to the licensee. See Format Privileges for details.

Field ID	Edit	Description
MAC_Address	Yes[1]	A MAC address for which the license file is valid. More than one entry may exist. Only supported on Windows™ and Linux.
Privileges	Yes[1]	Specific stamping privileges that are granted to the licensee. See Stamping Privileges for details.

[1] Value must be in the set that is specified in the license file.

[2] Value must be in the range that is specified in the license file.

Stamping Privileges

Each function in the stamping tool must be enabled by the TAC check before it can be used. If the stamping tool is run from within the Maconomy organization on a machine that satisfies the TAC environment check, all functions are enabled by default. Otherwise, privileges are disabled unless specifically granted by a license file.

Normally a function in the stamping tool can only be used if the corresponding privilege is in the license file. However, for the unscramble function, which operates on an already stamped file, an exception is made if the file stamp is a subset of the license, that is, if the stamp could have been created using the rights that are granted by the license file. This enables users to unscramble their own stamped files, as well as files that were stamped by developers within the same organization.

Privileges are represented as a bit field, where each bit represents one privilege. Different privileges can be combined using bit-wise disjunction (OR), and the resulting bit field is passed to the stamping tool as a hexadecimal (hex) number in the **Privileges** TAC field. The following table provides an overview of the available privileges and their bit-field values.

Because it is tedious and error-prone to combine bit-field hex values manually, keywords are defined for each privilege that can be used when assembling a set of privileges. The keywords that can be used are:

- The TAC field name that is associated with the privilege (See TAC Field Overview).
- The short or long form of the command-line argument that is associated with the privilege (see Command-Line Options Overview).
- `-l*` or `--login*` to set the “Global Login” privilege.

To assemble a set of privileges, the corresponding keywords can be combined in a comma-separated list as the argument to –-TAC Privileges (see TAC Field Overview) or they can be specified one by one in separate command-line arguments.

Example

To grant privileges to the options `-c`, `-a`, and `-u` you can use any of the following commands:

```
--TAC Privileges -c,-a,-u
--TAC Privileges --customernumber,--addons,--unscramble
--TAC Privileges CustomerNumber,AddOns,-u
--TAC Privileges 2,-a,40
--TAC Privileges -c --TAC Privileges AddOns,40
```

Privilege	Description	Hex	Bin
Add-on	Licensee is allowed to use the option `-a / --addons`.	0010	0000 0000 0001 0000
Create TAC	Licensee is allowed to use the option `-C / --CreateTAC`.	0008	0000 0000 0000 1000
Customer number	Licensee is allowed to use the option `-c / --customernumber`.	0002	0000 0000 0000 0010
Global login	Licensee is allowed to use the server wildcard '*' in login privileges. Privilege keyword: `-l* / --login*`.	0204	0000 0010 0000 0100
Login	Licensee is allowed to use the option `-l / --login`.	0004	0000 0000 0000 0100
Namespace	Licensee is allowed to specify the TAC field `Namespace`.	0100	0000 0001 0000 0000
No license	Licensee is allowed to use the option `-n / --nolicense`.	0020	0000 0000 0010 0000
Official	Licensee is allowed to use the option `-p / --package`.	0080	0000 0000 1000 0000
Unscramble	Licensee is allowed to use the option `-u / --unscramble`.	0040	0000 0000 0100 0000
Viewable	Licensee is allowed to use the option `-V / --viewable`.	0001	0000 0000 0000 0001

Format Privileges

For each file format that is supported by the stamping tool, a *format privilege* enables the licensee to read and write files of that format. The licensed formats are represented as a comma-separated list in the `FormatPrivileges` TAC field. The following table shows the supported formats.

Some format privileges overlap with the stamping privileges. For instance, the format privilege "License" implies the presence of the privilege "Create TAC."

Format	Description
License	TAC license file format.

Format	Description
Mscript	Full-featured MScript format.
MScript4	Stamp format compatible with MScript 4.0 and earlier. This format is supplied for use with legacy installations only, and the privilege to use this format is only distributed under special circumstances.
MRL	Maconomy Report Language format.
MUL	Maconomy Universe Language format.
MOL	Maconomy Object Language format.
MPL	Maconomy Print Language format.
UDL	Universe Definition Language format.
MQL	Maconomy Query Language format.
MEXL	Maconomy Entity Extension Language format.
MDEL	Maconomy Dialog Extension Language format.
DDL	Dialog Definition Language format.
XML	Unspecified XML format.
Config	Maconomy Server configuration files format(s).

Command-Line Options Overview

The stamping tool is invoked as:

```
MStamper (options) (file)*
```

One or more files can be specified on the command line after the options. If no files are specified, the program reads from standard input. Note that the options `-m`, `-u`, and `-s` cannot be used when reading from standard input. Supported options are listed in the following table.

Command-Line Options

Option	Description
`-a <addOnList>` `--addons <addOnList>`	Specifies that the stamped script should run on systems with all of the add-ons in `<addOnList>`, which is a comma-separated list of integer values. Used in license files, the legal groups of add-ons are restricted to those specified. Example: `'-a 64,65'` or `'-a 67'` *Required privilege: 10*
`-c <custNo>` `--customernumber <custNo>`	Specifies that the stamped script should run on systems with customer number `<custNo>`. Option `-c` can be used more than once. Used in license files, the legal customer numbers are restricted to those specified. *Required privilege: 2*
`-C` `--CreateTAC`	Create a TAC license file. Information from the current TAC license and from TAC fields specified on the command line is compiled into a new TAC license. Use the `-o` option to specify the name of the license file. *Required privilege: 8*
`-f <format>` `--format <format>`	Specify the stamp format to use. Valid format strings are: `MScript, MScript4` (default, no block modifiers); `MRL, MUL, MOL, MPL, MQL, UDL, MEXL, MDXL, DDL, Config` ('--' line prefix), `XML` ('<!--','-->' block comments). Example: `--format MRL` If a format is not specified, MStamper infers it from the input file's extension.
`-h` `--help`	Display help information.
`-L <filename>` `--license <filename>`	Specify the TAC license file. License information and access privileges are read from this file. The default is a license file named `license.tac` in the current working directory.

Option	Description
`-l <user>:<servers>` `--login <user>:<servers>`	Specifies a login privilege for the stamped script. The script can log in as the user `<user>` on the Maconomy servers that have the specified installation numbers (comma-separated list) without specifying a password. The wildcard token * can be used to match all servers. Used in license files, the legal login privileges are restricted to those specified. Example: `-l john:12345678,87654321` `-l bob:*` *Required privilege: 4 (+ 204 for '*')*
`-m` `--modify`	The input files are modified instead of sending output to a separate file. This option cannot be used if option `-o` is used. Note that either `-o` or `-m` must be used if `-u` or `-s` is not specified.
`-n` `--nolicense`	Specifies that the stamped script should run on any system. No special license is required. This option cannot be combined with `-c` or `-a`. *Required privilege: 20*
`-o <fileName>` `--output <fileName>`	Output is written to `<fileName>`. Otherwise, output is sent to standard output. This option cannot be used if more than one input file is specified, if option `-m` is used, or if `-O` is used, except when reading from the standard input.
`-O <dir>` `--OutputDir <dir>`	Output directory when processing files. This option cannot be used with option `-m` or if `-o` is used, except when reading from the standard input.
`-p` `--package`	Stamp a file as being an official M-Script package. Does not scramble the file. Use the option `--scramble` to turn scrambling on again. *Required privilege: 80*
`--pedantic`	Treat warnings as errors.
`-r` `--readable`	Stamp the file, but do not scramble it.

Option	Description
`--restamp`	This option enables the stamper to repair broken stamps. This can be especially useful with readable files that have been subjected to localization. To restamp a file you must have all of the privileges that are necessary to produce the stamp in the file. If the file is scrambled, you must also have the specific privilege to unscramble. *Required privilege: 40*
`--scramble`	If the option `--package` has been used to stamp a file, use this option to turn scrambling on again. The `--package` option does not scramble files, and it turns scrambling off.
`-s` `--strip`	This option implies option `-u` but also strips the licensing information from the file, thus returning it to its original form.
`-T <field> <value>` `--TAC <field> <value>`	Specify a value for a TAC field.
`-u` `--unscramble`	This option makes the stamping tool work in reverse mode, which means that a stamped file is unscrambled to a readable form. To unscramble a file, you must either have all of the privileges that are necessary to produce the stamp in the file, or the specific privilege to unscramble. *Required privilege: 40*
`-v` `--version`	Display version information.
`-vv` `--vversion`	Display extended version information.
`-V` `--viewable`	Make it possible to view the file unscrambled using the simple M-Script viewer. Note that the MScript viewer is deprecated, and so is this option. *Required privilege: 1*
`-x <key> <value>` `--extra <key> <value>`	Specify optional extra information to be stamped into the script. This information is readable in the stamped file. More than one pair of extra information is allowed. This is not backward-compatible with M-Script versions 1.0-4.0.

Examples

In the following examples we assume the identify of John Johnson. John is the head consultant of Johnson Consulting, Inc., which is a small but highly profitable consulting company due to its partnership agreement with Maconomy.

Johnson Consulting does a lot of customization for its clients, mostly M-Script programming in the Maconomy Portal. To ensure that the clients do not modify the solutions that they receive or even resell them, John wants to stamp all of the M-Script files that they distribute with the customer number of the client. John also has several consultants on staff, and he also wants to keep track of which files each of them produces.

From Maconomy, John has obtained a TAC license file that enables him to use the MStamper executable that was distributed with their Maconomy installation. The license allows him to specify customer numbers in the files that he is stamping, and also gives him the privilege to create individual license files for the members of his staff.

The command-line examples contain little or no explanation of the options used. See Command-Line Options Overview for more information.

Stamping a File

The most common use of the stamping tool is to stamp a file. Say that John has a list of files `file1...filen` that he wants to stamp with a specific customer number (custno). Assuming that the `license.tac` license file resides in the current directory, he can invoke the MStamper from a command line like this:

```
MStamper -c custno -m file1...filen
```

This will modify the files and stamp them with the specified customer numbers.

If the license file resides in some other directory or does not have the name `license.tac`, he can specify its location with the `-L` option:

```
MStamper -L dir\license.tac -c custno -m file1...filen
```

Because there is more than one input file, he cannot specify the output file. Had there been only one input file, he could have replaced the `-m` option with `-o` to write the stamped file to the directory `out`:

```
MStamper -L dir\license.tac -c custno -o out\file file
```

Creating a License File

John wants to create an individual license file for one of his consultant staff members, Jim James. Jim should be able to stamp M-scripts with customer numbers and to unscramble already stamped scripts. John must also remember to set the expiration date of the license, because the default is always “today.” By specifying an empty string, he can disable the expiration restriction:

```
MStamper -L dir\license.tac -C -T Author "Jim James"
   -T Privileges -c,-u -T FormatPrivileges MScript
   -T ExpiringDate "" -o license.jim.tac
```

Now John has created a license file for Jim called `license.jim.tac`. The author field has been changed to “Jim James,” so this will be the author information in all of the files that Jim stamps.

Suppose now that John rethinks his choice of license to give Jim: With a license that never expires, John will never be able to restrict Jim’s TAC privileges, because Jim can always just use a backup of his old license. John therefore decides to give Jim a license that will be valid until

New Year's Eve 2005. He also wants to restrict the license to be valid only on Jim's laptop, which he happens to know has the MAC address `00e0f8d8`.

```
MStamper -L dir\license.tac -C -T Author "Jim James"
   -T Privileges -c,-u -T FormatPrivileges MScript
   -T ExpiringDate "31.12.2005" -T MAC_Address 00e0f8d8
   -o license.jim.tac
```

John reconsiders once again: Jim might be tempted to use the license to do some work for the company's clients on his own time. Not very likely, but better safe than sorry. While John cannot completely prevent this, he can restrict the companies for which Jim is licensed to do development work. If Jim is doing work for only two clients in the next month, John can stamp the customer numbers of these clients into Jim's license file, thus forcing him to stamp every file to one of these clients. By making the license file expire at the end of the month, John can also ensure that Jim will not be able to do any work for these clients later on without obtaining a new license file.

```
MStamper -L dir\license.tac -C -T Author "Jim James"
   -T Privileges -c,-u -T FormatPrivileges MScript
   -T ExpiringDate "31.05.2005" -T MAC_Address 00e0f8d8
   -T CustomerNumber custno1 -T CustomerNumber custno2
   -o license.jim.tac
```

Here John used `-T CustomerNumber` to set the customer numbers. This is equivalent to the `-c` option, which is only retained for historic reasons.

More Examples

For more examples, see Stamping and Installing Universes and Universe Reports.

MBuilder Reference

Overview

This manual describes how to install reports, Analyzer reports, universes, entity and dialog extensions, and other custom enhancements to Maconomy on the Maconomy Server.

This manual describes MBuilder version 6.4.0.

The MBuilder tool, version 1.0.0, was introduced in TPU 43 (server 30).

To support custom-built enhancements to Maconomy, such as custom universes and universe reports, MBuilder was introduced to support the installation of such files on the server. The following languages are supported by MBuilder:

- **MUL** — Maconomy Universe Language
- **MRL** — Maconomy Reporting Language
- **MOL** — Maconomy Object Language
- **MEXL** — Maconomy Entity eXtension Language
- **MDXL** — Maconomy Dialog eXtension Language

The purpose of the MBuilder is to install universes (MUL files), reports (MRL files), report layouts (MPL files), report M-scripts (MS files), and Maconomy entity and dialog extensions (MEXL and MDXL files). In addition, functionality to add external relations to a Maconomy installation is provided using MOL files with MBuilder. For more information about the individual languages, please consult the references listed in “Additional Documentation.”

Using the MBuilder Tool

The MBuilder tool runs primarily in three modes: MRL report (and MPL report layout and report M-script) installation, MUL universe installation, and custom MOL relation creation.

General Options

A set of general options is available in MBuilder, and a few of them are described here. For a complete list of all options, see Command-Line Options. The mandatory option `--Maconomy[Dir]` takes as an argument the folder path of the installation, for example, `D:\Maconomy\MaconomyHomes\W_8_0` on Windows NT. MBuilder uses a set of character conversion tables that are contained in a standard Maconomy installation. If MBuilder is used from a machine that has no Maconomy installation, a folder to the necessary character conversion tables can be specified using the `--CCTablePath` option.

When installing customer-specific reports, universes, and so on, the option `--Customization[Level] Custom` is mandatory. Finally, the `--Path` option specifies the folder of the source files that are necessary in the installation or creation (that is, the MRL, MPL, MUL or MOL files).

Installing MRL Reports, MPL Report Layouts, and Mscripts

When MRL reports and possibly their corresponding MPL layouts and report Mscripts are developed (see the MRL Language Reference for more details), they must be installed on the Maconomy server using MBuilder.

Installation of one or more MRL reports is normally done using the `--Report[s]` option. This also installs the associated layouts and report M-scripts. The file names of a layout and report M-script must equal those of their associated report, except for file extensions. If, for example, you want to install the report `myReport.mrl`, the layout must be named `myReport.mpl`, and the report M-script must be named `myReport.ms`. You can exclude the reports, layouts, or the report M-scripts from being installed by using one of the two options `--MplOnly` (do not install the report and the report M-script) or `--MrlOnly` (do not install the layout and the report M-script).

If an MRL file is not present, you can install a layout anyway by using the option `--layout`. In this case, the file name of the layout must be equal to the file name of the installed report (except for the extension). If the file names are not identical, the option `--Report` must be used to define the full module name of the report, in addition to the option `--layout`, which still specifies the layout file name.

The reports are installed in the Maconomy system under the customization level Custom when the mandatory option `--Customization[Level]` Custom is used.

Before installation, the report, layout, and report layout files must be stamped using the MStamper tool. When stamping the files, the option `--readable` must be used for the MStamper. For more information, see the MStamper Reference.

Support for MRL 1.3, Links, and Report Collections

MRL version 1.3 introduced links and report collections.

If a report contains links, the option `--ValidateLinks` must be used whenever the links are to be validated. Sometimes, however, a report links to a report that is not yet installed (for example itself). In such situations, link validation fails, meaning that link validation should be disabled (which it is by default). To ensure valid links, run MBuilder with link validation enabled, after the report to which links are made has been installed.

Report collections may be associated with several layouts (but at most one report Mscript). MBuilder automatically finds the associated files. The strategy for associating layouts with a report is as follows: an MPL file is associated, provided that the first unqualified part of its file name matches the first unqualified part of the report file name. For example, if the report is called `a.mrl`, MBuilder associates all files in the present directory named `a.mpl, a.b.mpl, a.c.mpl, a.b.c.mpl`, and so on with the report, but ab.mpl or ab.c.mpl are not associated. As described in the MRL Language Reference, each layout must specify which member of the collection it is to be associated with.

Debug Information for Installed MRL Reports

MBuilder can return debug information for an installed MRL Report. The returned debug information contains a list of parameters for the report and the SQL statements that are used when the report is executed. Note that to get the debug information for a report, the report must be installed in advance on the Maconomy server.

The debug information is returned when the option `--DebugInfo <sqltype>` is used. The `<sqltype>` selects the type of SQL that is generated. The name of the installed report is given with the mandatory attribute `--ReportName <name>`.

The following example shows debug information for the installed report `dw::R001`.

Report, debug info

```
================================================================================
id : 'dw::R001'
title : 'Demo Report 001'
```

```
Parameters
--------------------------------------------------------------------------------
id : 'parmEmployeeNumbers'
title : 'Employees'
mandatory : 1
defaultValue : ''
SQL for cursor ''
--------------------------------------------------------------------------------
SELECT r1.EmployeeNumber AS EmployeeNumber,
r1.Name1 AS EmployeeName,
SUM(((((((r2.NumberOfDay1+r2.NumberOfDay2)+r2.NumberOfDay3)+r2.Num
berOfDay4)+r2.NumberOfDay5)+r2.NumberOfDay6)+r2.NumberOfDay7)) AS
NumberOfWeekSUM
FROM
ACEmployee r1, ACTimeSheetLine r2
WHERE r1.EmployeeNumber = r2.EmployeeNumber AND (1=1)
GROUP BY r1.EmployeeNumber,
r1.Name1
ORDER BY r1.EmployeeNumber ASC
```

Installing MUL Universes

As for MRL reports, MUL universes must be installed in the Maconomy installation. Installation of MUL universes is done using the --Universe[s] option. The universes are installed in the Maconomy system under the customization level Custom when the mandatory option `--Customization[Level] Custom` is used. When installing a universe, the source file is installed next to the runtime version of the file. The source file is not used by the server, but keeping the source file near the runtime file makes it easy to find if you need to modify it.

All MUL universes must be installed using MBuilder.

Before you install universes, the files must be stamped using the MStamper. When stamping the files, the options `--readable --format WBL` must be used for MStamper. For more information, see the MStamper Reference.

Installing Entity and Dialog Extensions

Entity extensions and dialog extensions are specified in MEXL and MDXL files, respectively, and are used for adding custom actions to entities (objects) in Maconomy and specifying how they are displayed in individual dialogs. For more information, see the Maconomy Extension Languages Reference.

MEXL

The MEXL source files can be named freely, but when they are installed they are given their entity name (possibly abbreviated). For example, MBuilder would rename the source file `myJobHeader.mexl`, which extends the `JobHeader` entity, to `jobheader.mexl` on the server. Only one MEXL file per entity can exist on each level of customization (that is `Standard`, `Solution`, and `Custom`).

M-Script source files must be located in the correct module structure. For example, if the source file `myJobHeader.mexl` contains the specification of an action with `method=myAction()` in `package=MScript::JobHeader::MyActions(1.2)`, the file `MScript\JobHeader\MyActions.1.2.ms` must exist relative to the location of the `myJobHeader.mexl` source file.

The `-ee` option installs MEXL and associated M-Script files on the server. To install the MEXL files only, add the option `-eo`.

MDXL

The MDXL source files can be named freely, but when they are installed they are given their dialog name (possibly abbreviated). For example, MBuilder would rename the source file `myJobs.mdxl`, which extends the `Jobs` dialog, to `jobs.mdxl` on the server. Only one MDXL file per dialog can exist on each level of customization (that is, `Standard`, `Solution`, and `Custom`).

The `-de` option installs MDXL files on the server.

Character Encoding: Using Non-Western Characters

Universes may use non-Western characters, such as Cyrillic. In these cases universes must be installed with the proper character encoding. Before running MBuilder, the environment variable

`MACONOMY_LCP_ENCODING`

must be set on the system to which the universes are installed. For example, if Cyrillic encoding is used on a Windows system, the value of the environment variable is `Windows-1251`.

Character encoding is specified in the initialization file of the Maconomy server. However, because MBuilder is independent of these settings, setting the `MACONOMY_LCP_ENCODING` variable in the server initialization file is not sufficient; that is, you must set both the initialization file and the environment variable.

Creating Custom MOL Relations

MOL—the Maconomy Object Language—is the language that is used for making external relations available in the Maconomy system. The external relations are accessible like other Maconomy objects (relations) and can, for instance, be incorporated in universe reports using the languages MQL and MUL.

Prerequisites

To facilitate the use of custom relations, add-on 103 (“Custom Database Extensions”) must be active on the installation. In addition, a license file that contains the customer-specific namespace must exist (the license file is supplied by Maconomy).

When the custom relations are installed on an installation (when the custom `DbDesc` file is copied to the `Database` folder of the Maconomy installation), they can be accessed (read-only) like any other Maconomy standard relation, for example, from MUL universes, M-Script, and so on.

DbDesc Creation

To create custom MOL relations, one or more MOL source files that contain object specifications must be prepared in advance. See the MOL Language Reference for more information about how to use MOL. The names of the MOL objects must be prefixed by the customer-specific namespace that is supplied by Maconomy. These files are passed to MBuilder using the `--Objects` option.

The MOL source files are compiled into a binary database description file, a `DbDesc` file. The compilation includes a namespace validation of all MOL objects and the namespace given in the license file. The license file is passed to MBuilder using the option `--LicenseFile`.

The following is an example of a MOL object using namespace `abc`.

```
<MOL 1>
   <Object abc_MySalary>
   .Componentid :String :Key+
   .Target :Integer
   .Salary :Amount
   .SalaryGroup :String :"Salary level"
   <End Object>
```

The name of the resulting binary DbDesc file is in the form

`DbDesc.<namespace>[.<subname>].xdd`

where `<namespace>` is the four-letter namespace string that is supplied by Maconomy, and `<subname>` is an optional naming string, for example:

`DbDesc.abc.MySalary1.xdd`

or without subname:

`DbDesc.abc.xdd`

The optional subname is specified using the option `--SubName`.

Installation of the new `DbDesc` file is done manually by copying the new `DbDesc` file to the server `MaconomyDir/Database` folder. Note that none of the existing `DbDesc` files (`DbDesc` and possibly `DbDesc.Portal`) may be deleted or renamed.

Creating the Database Views

The MOL specifications only constitute the Maconomy side of the process, that is, the specification in Maconomy terms of fields, types, and attributes. For every MOL object, a corresponding database table or view must be created manually. It is essential that the MOL objects and their database table counterparts correspond in field definitions, that is, field order, types, and so on. See the next section for more information about the Maconomy type system's correspondence with database table types. The actual implementation of the database table is highly database-specific and must be done manually. To gain access to remote databases, the database table can be implemented as a database view that references external data.

Maconomy Types vs. Database Types

When using the MOL language, all field definitions must adhere to the Maconomy Type System; that is, all MOL fields must consist of one of the predefined Maconomy types. The custom-created database relations or views that correspond to each MOL object must likewise obey a set of Maconomy Type System conversion rules, depending on the individual database installation. In the following table, all of the supported Maconomy types are listed with their specific database counterparts. This means that, for example, the MOL field **myDate** of Maconomy type `AMOUNT` must be implemented as `NUMBER(20,2)` in an Oracle database.

Type Conversion Table					
Maconomy	**Null Value**	**Formal Notes**	**Oracle**	**SQL Server**	**DB2**
INTEGER	0		INTEGER	INTEGER	INTEGER
REAL	0		REAL	FLOA	FLOAT
AMOUNT	0		NUMBER(20,2)	FLOAT	DECIMAL(20,2)
BOOLEAN	0	[0, 1]	INTEGER	INTEGER	INTEGER
STRING	' '(one blank)	String max. 255 chars.	VARCHAR(N1)	VARCHAR(N1)	VARCHAR(N1)
DATE	' ' (one blank)	yyyy.mm.dd	VARCHAR(12)	VARCHAR(12)	VARCHAR(12)
TIME	' ' (one blank)	hh:mm:ss	VARCHAR(8)	VARCHAR(8)	VARCHAR(8)
<Enumeration>	-1	<Enumvalues>	INTEGER	INTEGER	INTEGER

The sizes of the database string fields depend heavily on the individual database installation. The default (and maximum) size is 255, but if string fields are included in database indexes, special limitations on the total size of all index fields might exist for the individual database installation.

Example

In an Oracle database, the fields in the MOL object `abc_MySalary` below... `<MOL 1>` `<Object abc_MySalary>` `.Componentid :String :Key+` `.Target :Integer` `.Salary :Amount` `.SalaryGroup :String :" Salary level"` `<End Object>`	...must be implemented as fields of the following types: `Componentid VARCHAR(255)` `Target INTEGER` `Salary NUMBER(20,2)` `SalaryGroup VARCHAR(255)`

Viewing DbDesc Contents

The DbDesc files are binary files and cannot be viewed. However, MBuilder offers functionality to dump the contents of a specified custom `DbDesc` file in text format. Using the option `--Dump`, a custom `DbDesc` file can be specified, and the contents will be dumped to a text file. The name of the text dump file is similar to the `DbDesc` file, but substituting the extension .xdd for .txt. For instance, the dump file for the file `DbDesc.abc.MySalary1.xdd` is called `DbDesc.abc.MySalary1.txt`. The dump file contains all relations, fields, types, and so on.

Converting Analyzer Files

To run the Analyzer in the Windows client, the compiled Analyzer files that worked for the old (pre-4) version of the Analyzer must be converted to a new format. This is done by first running the MBuilder, which creates universes and relations, and then running the server with a specific option, which installs the views.

Converting Files

All compiled Aalyzer files (`.grf`) should be located in the `Analyze` folder in the `MaconomyDir` on the server.

Files are converted by running MBuilder with the option `--Analyzer`.

When the MBuilder tool has been run without errors, you must run the server with certain command-line options to install views. Execute the following command on the server as the “maconomy”user:

```
MaconomyServer.<appshortname> -S<shortname> --AnalyzerViews
```

For instance:

```
MBuilder --Analyzer --MaconomyDir /data/maconomy/w_8_0
MaconomyServer.w_8_0 -Sw80 --AnalyzerViews
```

The commands are the same on Windows NT and Unix.

Command-Line Options

MBuilder can be invoked in four modes: install reports and/or report layouts and/or report M-scripts, install universes, create `DbDesc`, and finally dump `DbDesc` content information.

Overview

Some of the command-line options are generic and can be applied in all modes, and some options are specific to the selected mode. The notes that are referenced in the “Note” column are listed after the tables in this section.

Generic Options

Option	Argument	Note	Description
`-h` `--Help`	`<boolean>`		Display the help text.
`-v` `--Version`			Display the current program version.
`-m` `--Maconomy[Dir]`	`<folder>`	(M)	Select Maconomy folder.
`-cc` `--CCTablePath`	`<folder>`		Select special `CCTable` folder that contains character conversion table files. Used if no Maconomy installations exists.

Option	Argument	Note	Description
-c -- Customization[Level]	<cust-level>		Which customization level to install in (default: Custom).
-p --Path	<folder>		Folder for directory that contains files to use (default: .)
-nb --NoBackup	<boolean>		Do not make backup of files overwritten by the installer (default: false).
-ow --Overwrite	<boolean>		Allow overwrite of existing installed files (reports and dialog extensions) (default: false).
-ef --ErrorFile	<filepath>		Full path for error file. stderr is used if not specified.
-lf --LogFile	<filepath>		Full path for log file. stdout is used if not specified.
-hl --HaltLevel	<errorlevel>		Define errorlevel (and higher) on which the program halts (default: Error).
-el --ErrorLogLevel	<errorlevel>		Define errorlevel (and higher) on which the program logs (default: Warning).
-a --Validation	<level>		Define level for semantic validation (default: Complete).
-b --Verbose	<verboselevel>		Log data during run. Enable several options by adding ids (default: 0).
-s --Simulate	<boolean>		Do not perform the installation, simulate it. Only perform the checking (default: false).

Report, Report Layout, and Report M-Script Installation Options

Option	Argument	Note	Description
-r --Report[s]	<filename(s)>	(1,2)	Name(s) for report(s), layout(s) and report M-script(s) to install.
-l --Layout[s]	<layoutname(s)>	(1,2)	Layoutname(s) for layout(s) to install.
-ro --MrlOnly	<boolean>		Only install the MRL report file (default: false).

Option	Argument	Note	Description
-po --MplOnly	<boolean>		Only install the MPL layout file (default: false).
-di --DebugInfo	<sqltype>	(1,3)	Dump debug information for an installed report.
-rn --ReportName	<modulename>	(3)	Name of an installed report.
-vl --ValidateLinks	<boolean>		Perform validation of links (default: false).

Universe Installation Options

Option	Argument	Note	Description
-u --Universe[s]	<filename(s)>	(1)	Compile and install the specified MUL file(s).

Entity Installation Options

Option	Argument	Note	Description
-ee -- EntityExt[ensions]	<filename(s)>		Install the specified MEXL file(s), including any M-scripts referenced in package attributes.
-eo --MexlOnly	<boolean>		Install only the MEXL file(s), not the M-Script files (default: false). This option is appended to the line (usage: -ee <file> -eo)

Dialog Installation Options

Option	Argument	Note	Description
-de -- DialogExt[ensions]	<filename(s)>		Install the specified MDXL file(s).

Custom Relation Creation Options

Option	Argument	Note	Description
-o --Object[s]	<filename(s)>	(1)	Create a DbDesc file (for example DbDesc.namespace.xdd) that contains data from the specified MOL files.

Option	Argument	Note	Description
-li --LicenseFile	<filepath>		File path to specific license TAC file including namespace (default ./License.tac).
-sn --SubName	<name>		Subname for the DbDesc file (for example, DbDesc.namespace.subname.xdd) (default: empty).
-op --OutputPath	<filepath>		Define outputfolder for the DbDesc file (default: .).

Custom Relation Dump Options

Option	Argument	Note	Description
-d --Dump	<filepath>	(1)	Dump relation documentation for specified custom DbDesc file. Output name is the same as the input name with a txt extension.

Analyzer Conversion Options

Option	Argument	Note	Description
-z --Analyzer		(1,4)	Convert all Analyzer (.grf) files in the Analyze directory in MaconomyDir

Notes

- **(M)** The option is mandatory.
- **(1)** One of either --Report and/or --Layout or --Object or --Universe or -Dump or --Analyzer or --Dialog or --DebugInfo must be specified.
- **(2)** If only --Report is specified, MRL, MPL and MS files (if they exist) are installed.

  If only --Layout is specified, only the MPL file(s) is installed. The filename of the MPL file must be identical to the filename in the layout name

  If both --Report and --Layout are specified, only the MPL file(s) are installed. The filename of the MPL file(s) is given by --Report (same order)
- **(3)** If --DebugInfo is specified then --ReportName is mandatory
- **(4)** When running --Analyzer, the only generic options available are verbose and Maconomy(Dir)

Types

The following tables specify the available types for the MBuilder options.

Generic Option Types

Option Type	Values	Description
<boolean>	{Empty}=Yes | Yes | No | True | False	
<errorlevel>:	System | Fatal | Error | Warning | Notify	Internal system error. Syntax error and so on. Major semantic error in specification (system unable to run). Minor semantic error in specification (system is (partly) runable). Non-optimal specification.
<filename(s)>	{File name}* | All	Specified files. Use all files of right type in given directory.
<modulename(s)>	{Module name of report}*	Example: Module::Submodule::MyReport.
<cust-level>	Custom | Solution | Standard	Install as Custom extension. Install as Solution extension (not available). Install as Maconomy extension (not available).
<level>	Syntax | Complete	Syntax checks only. Syntax and semantic checks.

Report and Report Layout Installation Types

Option Type	Values	Description
<verboselevel>	[0..15]	Bitmask, consists of the following values: 0 – No output (except errors). 1 – Parsed file name(s). 2 – Dump output file name(s). 4 – Error count. 8 – Dump parsed file data (as tags and attributes).

Option Type	Values	Description
<sqltype>	Oracle | MSQL | DB2 | DebugOracle | DebugMSQL | DebugDB2 | Debug	SQL for Oracle is generated. SQL for Microsoft SQL-Server generated. SQL for IBM DB2 generated. SQL for Oracle is generated in nice format. SQL for Microsoft SQL-Server generated in nice format. SQL for IBM DB2 generated in nice format. Special debug SQL in nice format.

Universe Installation Types

Option Type	Values	Description
<verboselevel>	[0..63]	Bitmask, consists of the following values: 0 - No output (except errors). 1 - Parsed file name(s). 2 - Error count. 4 - Dump parsed file data (as tags and attributes). 8 - Dump interface to universe. 16 - Dump testsuite data. 32 - Dump dependency data.

Dialog Extension Installation Types

Option Type	Values	Description
<verboselevel>	[0..15]	Bitmask, consists of the following values: 0 - No output (execpt errors). 1 - Parsed file name(s). 2 - Dump output file name(s). 4 - Error counts. 8 - Dump parsed file data (as tags and attributes).

Custom Relation Creation Types

Option Type	Values	Description
<verboselevel>	[0..1]	Bitmask, consists of the following values: 0 - No output (execpt errors). 1 - Progress info.

Additional Documentation

The following set of related documentation is available:

- Getting Started with Universe Reports
- MRL Language Reference
- MUL Language Reference
- MOL Language Reference

- MQL Language Reference
- Maconomy Extension Languages Reference
- Stamping and Installing Universes and Universe Reports

Version History

MBuilder 6.3.0

Support for installing MXEL and MDXL files added.

MBuilder 6.0.0

Support for installing report collections with multiple layouts added. Option `--ValidateLinks` added.

MBuilder 5.0.0

Support for installing Report M-scripts added.

MBuilder 4.0.0

Option `--overwrite` added.

MBuilder 3.0.1

Added debug information for installed MRL Report.

MBuilder 1.0.0

Initial version.Text

MDumper Reference

Overview

This manual describes how to extract, or dump, information from a Maconomy installation. The information that is dumped by MDumper can be viewed as HTML files using MDoc, which is a part of the MDumper.

This manual describes MDumper version 1.2.7.

A Maconomy installation contains a huge amount of information. To gain access to some of this information, extracts must be made in specific formats to use it properly. MDumper is the tool to extract, or dump, information from a Maconomy installation. The information that is dumped by MDumper is used in various independent ways.

Some of the installation information is valuable knowledge when developing customized extensions to Maconomy, such as universes and universe reports. To support easy access to this information, MDumper features the concept of MDoc. MDoc is an HTML-based dump of basic structures of the Maconomy installation.

The MDumper tool version 1.2.0 was introduced in TPU 50.

Using MDumper

The MDumper tool can be invoked in several dump *modes*. The important modes are the *MDoc* dump mode and the *Snapshot Definition* dump mode.

Note that if MDumper is invoked on Linux/Unix platforms, the user who executes the tool must have read/write access to the `MaconomyDir` folder.

General Options

A set of general options is available in MDumper, and a few of them are described here. The mandatory option `-Maconomy` takes as an argument the folder path of the installation, for example `D:\MaconomyNT\MaconomyHomes\W_8_0` on Windows servers. The Maconomy option can also be called `-MaconomyDir or -MaconomyFolder`.

MDumper uses a set of character conversion tables that are contained in a standard Maconomy installation. If MDumper is used from a machine that has no Maconomy installation, a folder to the necessary character conversion tables must be specified using the `-CCTableFolder` option.

Dump Options

To auto-generate MDoc information, use the option `-DumpMDoc`.

To dump snapshot definitions, use the option `-DumpSnapshotDefs`.

Creating MDoc Information

Using MDumper v. 1.2.0 and later, you can create an HTML-based information dump of essential Maconomy objects.

When developing and using Maconomy universes and universe reports, it is important to know the available objects in the Maconomy system, for example database objects (relations), universe interfaces, and internal field types.

Auto-generated documentation of these basic Maconomy structures that is dumped by MDumper is called MDoc. MDoc is an entire web site that consists of a set of HTML files. Database objects (relations), field types, and universe interfaces are described in the MDoc system.

MDoc is created using the MDumper option `-DumpMDoc <destfolder>` where `<destfolder>` is an existing destination folder. For example, on Windows the command could look like this:

```
MDumper –MaconomyFolder c:\maconomy\w_8_0 –DumpMDoc c:\MDoc
```

Running MDumper with the `-DumpMDoc` option extracts a huge amount of information from the specified Maconomy installation, and consequently the actual execution of MDumper may take some time.

The MDoc dump is a static view of the system as it looks at the time of the generation of the MDoc dump. This means that after the installation of new universes, and so on, the MDoc web site must be regenerated to reflect the current Maconomy installation. Information that is only available in the database (such as dynamic type enumerations) is not accessible using MDoc.

After the dump is completed, the MDoc destination folder structure looks similar to the following example (only the `MDoc/html` folder is expanded):

```
MDoc
+-- html
| +-- relations
| +-- universeinterfaces
| +-- styles
| +-- index.html
|
+-- installation
+-- xml
```

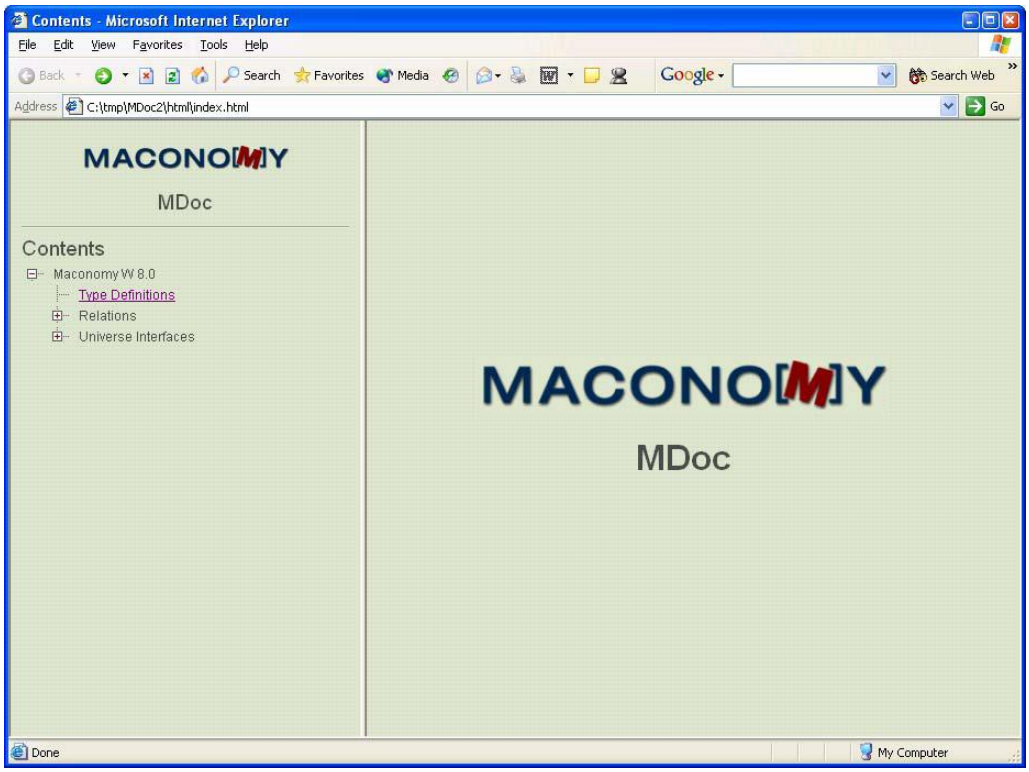
All information is dumped as structured XML data in the xml subfolder. After the XML files are dumped, they are translated into HTML using the XSLT style sheets found in the `installation` folder.

All MDoc information is based on client-side scripting, and can thus be accessed without the use of a web server. To view the MDoc information, open the file

```
MDoc/html/index.html
```

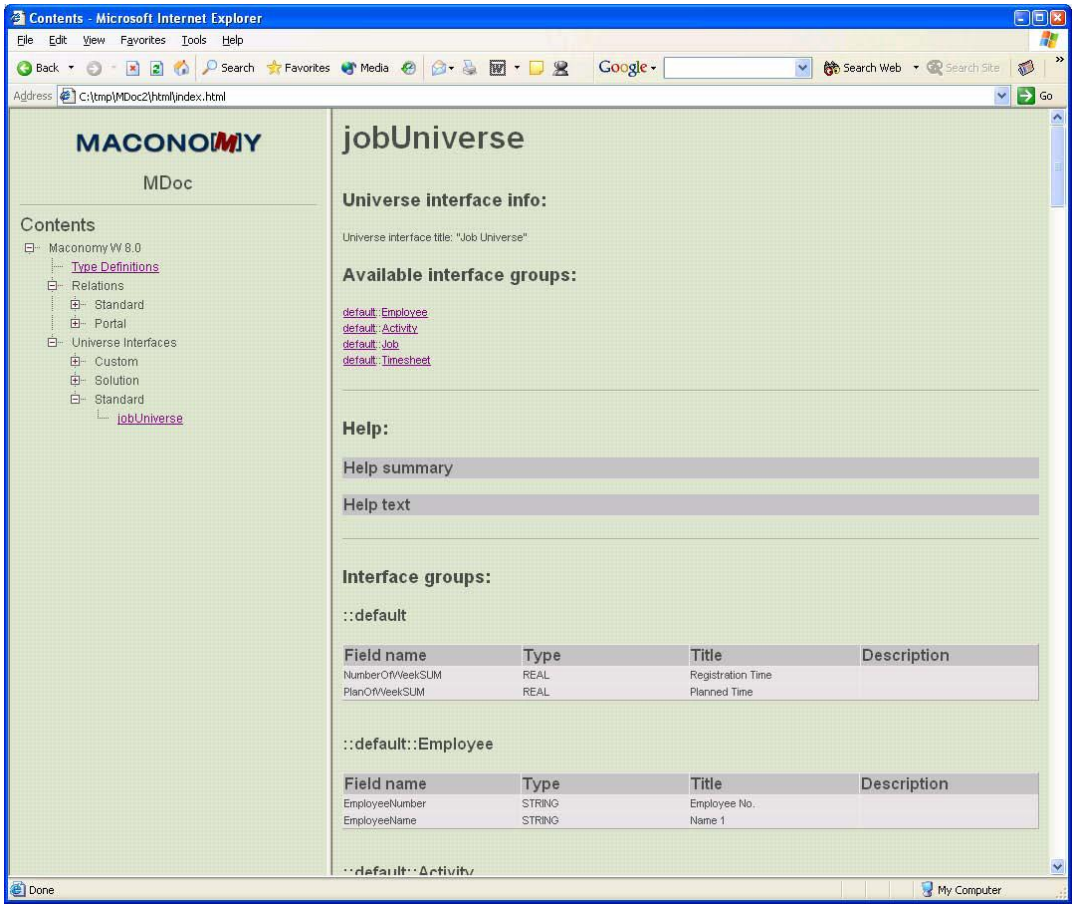
in a browser. Note that the MDoc web site is dependent on the use of JavaScript, so the browser must be JavaScript-enabled.

When starting MDoc in a browser, you see the following after you expand the `Maconomy` entry.



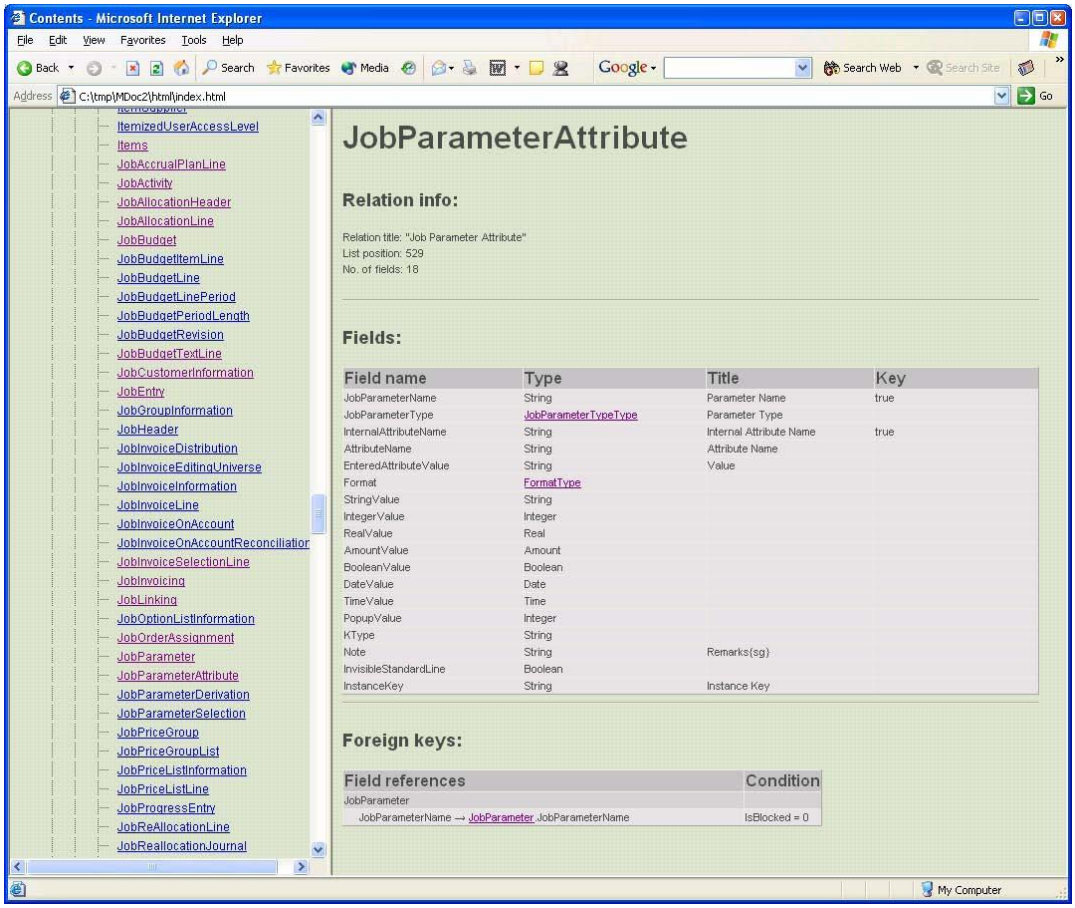
The page is divided into two panes. The left-hand pane is a tree view of all of the available information. Selecting links in the tree view shows the corresponding description page in the right-hand pane. The description page depends on the kind of object that is shown.

If you choose a universe Interface, the resulting window could look like the following.



The description page shows information about the selected universe such as title, interface groups, and their fields.

Relation description pages show information about title, fields, and foreign key references.



Command-Line Options

Overview

Some of the command-line options are generic and can be applied in all dump modes, and some options are specific to the selected mode.

Generic Options

Option	Argument	Note	Description
`-h` `-Help`	`<boolean>`		Display the MDumper help text.
`-v` `-Version`			Display current program version.
`-vv` `-VVersion`			Display current program version in extended format.
`-m` `-Maconomy[Folder|Dir]`	`<folder>`	(M)	Select Maconomy folder (for example, `/maconomy/w_8_0`).

Option	Argument	Note	Description
-CCTableFolder	<folder>		Select special CC_Table folder containing character conversion table files. Used if no Maconomy installation exist.
-OutputFolder	<folder>		Destination folder for MDumper output.
-SpecificDbDesc	<filepath>		Full path to specific DbDesc file to be dumped from.
-DumpRelationTypes	<enumlist>		Force relations of specific types to be dumped. <enumlist> contains a space-delimited list of one or more of the following options: [Standard | Expired | Network | Portal | System | Application | Universe | Custom | Analyzer]+.
-ExcludeRelationTypes	<enumlist>		When -DumpRelationTypes is used, -ExcludeRelationTypes rejects a list of relations of specific types. <enumlist> contains a space-delimited list of one or more of the following options: [Standard | Expired | Network | Portal | System | Application | Universe | Custom | Analyzer]+.
-DumpIndexList			Dump index list.
-DumpMDoc	<folder>		Generate an MDoc dump of the Maconomy installation into the specified output folder.
-DumpSnapshotDefs	<modulename>		Dump snapshot definition information. <modulename> is the name of an installed Universe Report, such as Module::Submodule::MyReport

Option	Argument	Note	Description
-Verbose	<verboselevel>		Log data during run. Enable process information output by increasing level of information (0-5): 0=No output 1=All errors ... 5=All output. The default is 0.

DumpIndexList Options

Option	Argument	Note	Description
-DumpIndexList		(1)	Internal option

DumpMDoc Options

Option	Argument	Note	Description
-DumpMDoc	<folder>	(1)	Generate an MDoc dump of the Maconomy installation into the specified output folder.

DumpSnapshotDefs Options

Option	Argument	Note	Description
DumpSnapshotDefs	<modulename>	(1)	Dump snapshot definition information. <modulename> is the name of an installed universe report, such as Module::Submodule::MyReport
-SQLTypes	<sqltypes>		Create a DbDesc (such as DbDesc.namespace.xdd) that contains data from the specified MOL files.
-Namespace	<name>		Filepath to specific license tac file including namespace (default './License.tac').

Notes

- **(M)** The option is mandatory.
- **(1)** One of -DumpIndexList, -DumpMDoc, or -DumpSnapshotDefs must be specified.

Additional Documentation

The following related documentation is available.

- Getting Started with Universe Reports
- MUL Language Reference
- MRL Language Reference
- MOL Language Reference

Stamping and Installing Universes and Universe Reports

Overview

This brief how-to guide describes the steps that are involved in stamping and installing universes and universe reports.

It is intended for graduates from the Maconomy Reporting Developer certification course.

You have been trained as a Maconomy Certified Reporting Developer, and you have received your license file. Now you want to test your knowledge and create, stamp, and install your own universes, universe reports, and layouts.

At the training, it was easy: Just press a keyboard shortcut in the Crimson editor, and everything worked nicely. However, now you must set up your own production environment.

Even though the MStamper and MBuilder manuals help you to set up this environment, this document helps you to get started quickly. The necessary options for stamping and installing are described, and suggestions are made to make stamping and installing a straightforward process.

Stamping — A Synopsis

A file stamp is a block of data that is inserted into the top or near the top of the file that is being stamped. The stamp consists of the following main components:

- **Message digest** — The checksum protection of the file, along with internal header information.
- **Tracking information** — Author and organization fields, creation date and time, and a history field, which traces the origin of the license used to stamp the file.
- **License information** — This information can be used to restrict the distribution of stamped files.

When a file has been stamped, it is ready to be transferred to the server and subsequently installed using the MBuilder tool.

The MStamper only works on computers that run Windows or Linux. This is because the computer's MAC address, which is required for a license file, cannot be determined on a Unix system such as AIX.

When transferring a stamped file from the Windows platform to a Unix server, it is important that you transfer it in binary format. If the file is transferred as text, you will be unable to install it on the server.

Installing — A Synopsis

MBuilder is the tool that is used for installing universes (MUL files), reports (MRL files), and report layouts (MPL files) on the Maconomy server. In addition, MOL files can be used with MBuilder for adding external relations to a Maconomy installation.

To run the installed reports in the portal, report portal components that refer to the reports must be created with the portal designer, and the portal should be configured to allow relevant users and roles to run the reports.

The installation must take place on the server. You must have access to the Maconomy server to install reports and universes.

Note for Unix Users

Because you cannot stamp files on a Unix machine, and because stamped reports must be installed using the MBuilder program on the machine that runs the Maconomy server, developers who use Windows or Linux for development and a Unix machine for running the Maconomy server should perform the following steps:

1. Stamp the file on the Windows/Linux development machine.
2. Transfer the stamped file to the Unix machine that runs the Maconomy server using FTP or SCP. Remember to transfer the file in binary format.
3. Run MBuilder on the Unix machine to install the report. This can be done remotely using Telnet or SSH.

Stamping and Installing

The stamping tool relies on the Maconomy Tools Access Control (TAC) system. This basically means that outside the Maconomy R&D department, a license must be provided for the tool to work. This license is obtained as a *TAC license file*. These license files can be used to grant the stamping tool certain rights, thus making it possible to distribute the tool to consultants, partners, and customers. This document describes how to stamp and install your own work.

Stamping

When stamping a file such as an MRL file, use the following command:

```
mstamper --readable --license <path to license file> --output <path to
    output directory> <file>
```

Deltek recommends placing a stamped file in a folder of its own, named by the current folder plus the extension .stamped. For example, if you are working on a report called `CompletionByDept.mrl in the folder \MyCustomer\Reports\Completion`, the stamped file should be placed in the directory `\MyCustomer\Reports\Completion.stamped`. This makes it easy to distinguish between source code and stamped files.

Installing

MRL report files and MUL files must be installed on the server using the MBuilder tool. When installing an MRL file, the MBuilder tool also installs an MPL file (a layout) with the same base filename, if it exists.

To install an MRL file (and possibly a layout), use the following command:

```
builder --MaconomyDir "<Application Home>" --customization Custom
   --Report <stamped file>
```

The --customization option must be set to Custom.

In the case of a MUL file, the option --Report must be exchanged with --Universe.

Helpful Tips

Instead of running the previously described commands from the command line every time that you need to stamp a file, you can automate the process using your development environment. The examples in this section use the Crimson editor as an example of how to automate the stamping and installation of a file on the server by assigning a batch file to a hotkey. Similar functionality is available in most other code editors, such as TextPad or the Visual C++ editor.

On the Maconomy Developer web site, you can find two scripts for stamping and installing files on the server. One script is for Windows; the other is for Unix. This section describes how to install the Windows script for use in the Crimson editor. Note that for the script to work, the editor must reside on the server machine.

The Maconomy Developer web site also contains syntax color-coding files for Crimson and EditPlus.

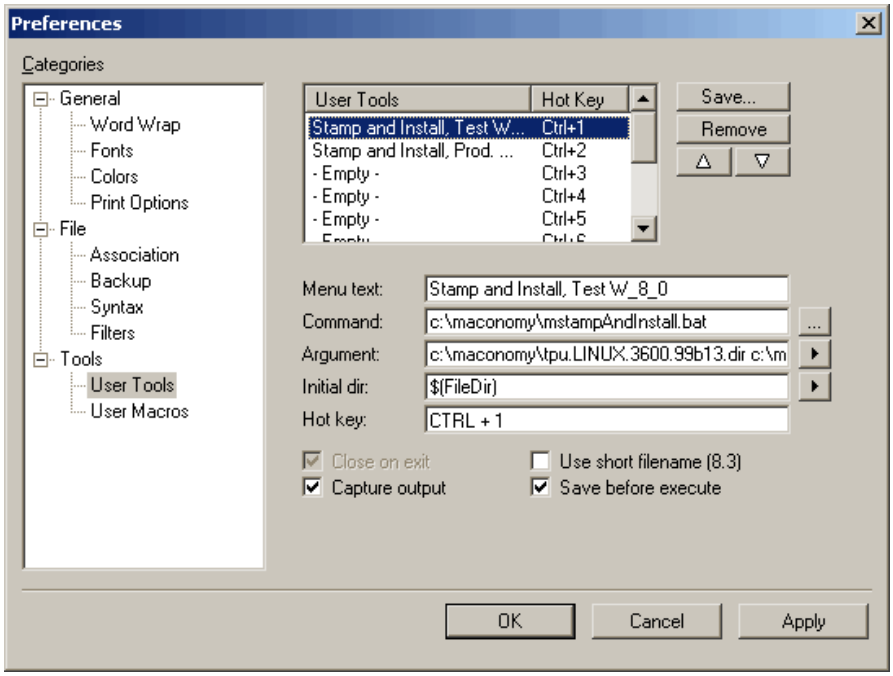
Batch Script for Stamping and Installing

Download the script `mstampAndInstall.bat` from the Developer web site (go to Technology » Tools » Miscellaneous). Note that the script was created with Windows 2000, and it may not work with your version of Windows.

Place the script on the server, for instance in `c:\maconomy`.

Set Up Crimson

To run the script file from within the Crimson editor, access the **Tools » Conf. User Tools** window. The following figure shows how to install the batch file in Crimson. Some comments are added after the figure.



Remember to save the setup by clicking “Save…”.

You can place the batch file `mstampAndInstall.bat` in any folder on the server.

Remember to enter the current directory, specified using $(FileDir) in Crimson, in the **Initial dir:** field.

Be sure to select the **Capture output** field so that you can see the output from the command after it has run. Otherwise, you may not be able to detect errors.

The idea behind doing it this way is that is that you can use the same batch script to support the installation of files in multiple locations. This is achieved by supplying arguments in the **Argument** field. As the **Menu text** field states, this tool (invoked by pressing **Ctrl+1**) stamps the current file and installs it in a test environment. The location of the test environment is specified in the **Argument** field. The hotkey **Ctrl+2** may use the same script to stamp and install the file in a production environment.

The script takes the following arguments:

- TPU folder, for instance `c:\maconomy\tpu.LINUX.3600.99b13.dir`
- APU folder, for instance `c:\maconomy\w_8_0`
- License file, for instance `c:\maconomy\license.tac`
- File name, for instance `c:\source\JobAnalysisU.mul`, specified using `$(FileName)` in Crimson. Make sure that the filename does not contain spaces, which is not guaranteed to work.

When you click **OK**, the “Stamp and Install” tool is ready to use when you press **Ctrl+1**.

Most editors, including TextPad and the Visual Studio editor, can be set up in a similar way. See the documentation for your particular product.

Deltek is the leading global provider of enterprise software and information solutions for professional services firms, government contractors, and government agencies. For decades, we have delivered actionable insight that empowers our customers to unlock their business potential. 20,000 organizations and millions of users in approximately over 80 countries around the world rely on Deltek to research and identify opportunities, win new business, optimize resource, streamline operations, and deliver more profitable projects. Deltek – Know more. Do more.®

deltek.com