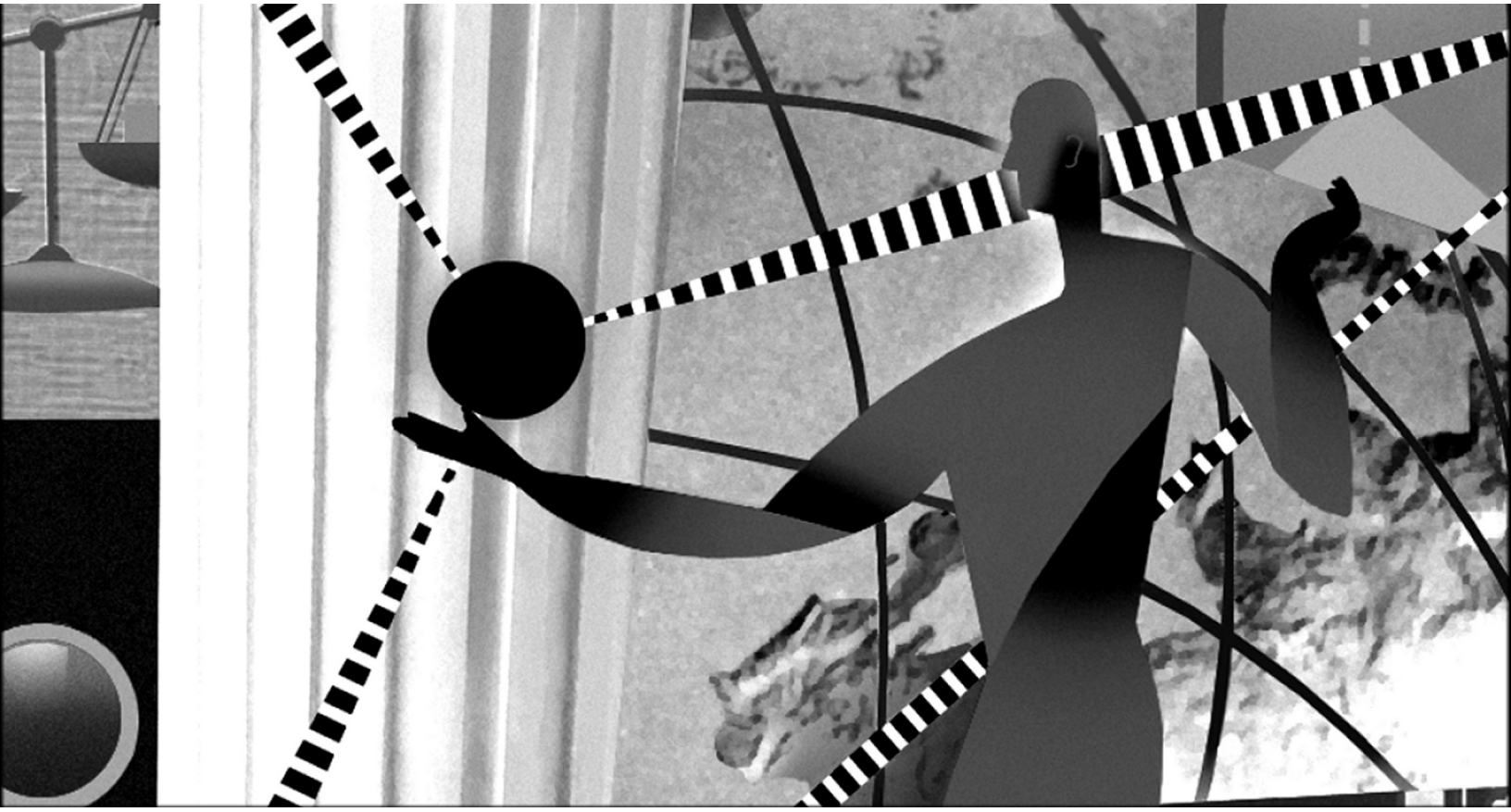
OASYS GLOBAL™ DIRECT



OASYS GLOBAL *DIRECT* MT511 PARSER API PROGRAMMER'S GUIDE AND REFERENCE

Version 3.4.2

21 February 2001

THOMSON FINANCIAL

esg

This document contains information proprietary to Thomson Financial, and may not be reproduced, disclosed or used in whole or in part without the express written permission of Thomson Financial.

Copyright © 2001 Thomson Financial, Inc. All rights reserved. No part of this work may be reproduced or copied in any form or by any means — graphic, electronic, or mechanical, including photocopying, recording, taping, or information and retrieval systems — without express prior written permission from the publisher.

All releases of OASYS Global™ *Direct*, the documentation and all other related materials are Thomson Financial's confidential and proprietary information and trade secrets, whether or not any portion thereof is or may be copyrighted or patented. All necessary steps must be taken to protect OASYS Global *Direct* and related materials from disclosure to any other person, firm or corporation, without the express written consent of Thomson Financial in each instance.

ALERT is a registered trademark in the U.S. and U.K. and used herein under license by Thomson Financial.

OASYS™ and OASYS Global™ are trademarks used herein under license by Thomson Financial.

Microsoft, Windows NT and other Microsoft products referenced are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Thomson Financial ESG
22 Thomson Place
Boston, MA 02210.

Printing: 2001

OASYS Global *Direct* MT511 Parser API Programmer's Guide and Reference

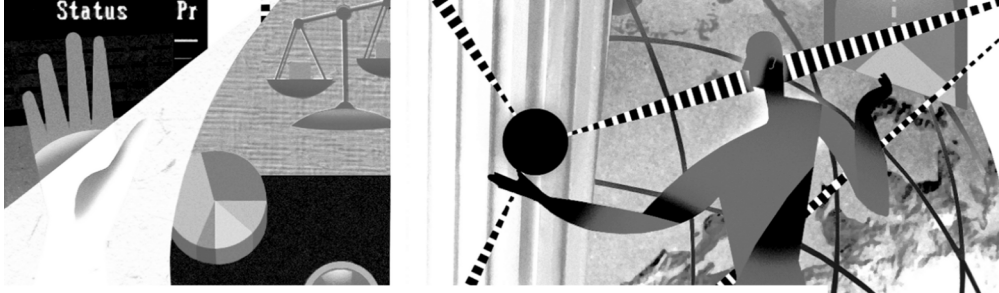
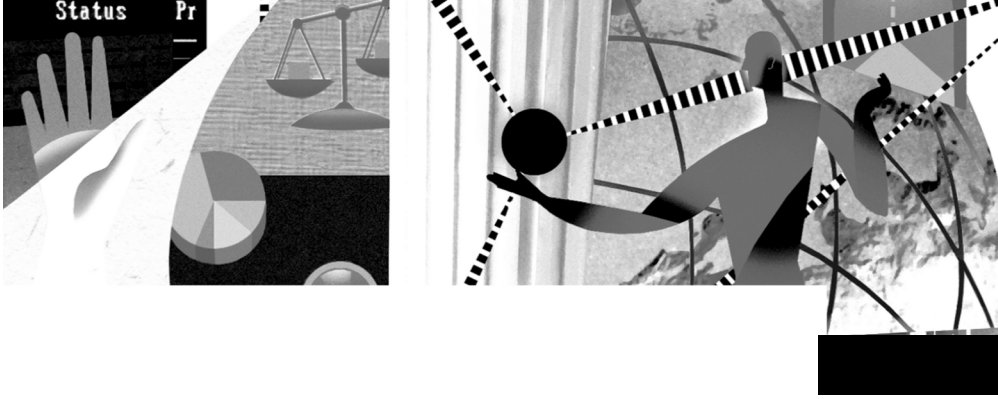


Table of Contents



Preface

This guide contains information on using the MT511 parser Application Program Interface (API) for OASYS Global *Direct* (OGD).

Intended Audience

This document is directed toward your systems analysts, and programmers, and others involved in implementing the link between your internal systems and OGD.

How This Manual Is Organized

This manual contains the following chapters and appendices:

- Chapter 1, “Introduction,” provides a brief survey of the current OASYS Global system by showing the message flow and the MT511 message protocol use for the new Direct host-to-host connection.
- Chapter 2, “Definitions,” defines terms related to OGD message processing.
- Chapter 3, “MT511 Parser API Functions,” provides detailed information about MT511 parser API functions.
- Chapter 4, “Sample Processing,” contains sample algorithms for processing received MT511 messages as well as creating new MT511 messages.
- Appendix A, “Validation Rules,” lists the rules that govern the process of MT511 message validation.
- Appendix B, “MT511 Parser Configuration Parameters,” lists the parameters used to configure the MT511 parser.
- Appendix C, “MT511 Tag to Parser Defined Constant Mapping,” lists the parser MT511 field defined constants `fieldId` as well as the field length defined constants associated with the MT511 tags.
- Appendix D, “MT511 Parser Installation Instructions,” provides MT511 parser installation information.

Typographic Conventions

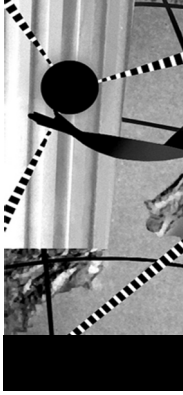
Unless otherwise noted in the text, this manual uses the following typographic conventions:

`Monospace`	Commands, printed text examples, function names and parameters, constants, variables, field names, literal values, return values, arguments, transaction names, configuration parameters, default values, format strings, MT511 tags and assigned values, path variables and paths, and C code samples; for example: `OASYS LOG REPORT`
`Monospace Bold`	Data format specifications. For example, **`dd-mmm-yy`** date format.
UPPERCASE	Electronic Trade Confirmation acronyms (such as ETC and API), and message types (such as BLIM).
Italics	Trade and message statuses (for example *Reject*, *Affirm*, *Cancel*).
UPPERCASE ITALICS	MT511 message types (for example, *AE*, *CN* (*CNA CNB*), and *TA*), and return codes (for example, *SUCCESS*, *FAILURE*).
Bold	File names (such as **import.dat** and **trans.map**), and library names (such as **wsock32.dll** and **moa.lib)**.

Related Documents

These are other Thomson ESG documents related to this publication:

- *OASYS Global Direct MT511 Messaging Specification*
- *OASYS Global Direct Broker and Institution Conformance Requirements*
- *OASYS Global Direct Message Delivery System TCP/IP API Programmer's Guide*
- *OASYS Global Direct Migration Guide for OASYS Global Automated Workstation Clients*
- *OASYS Global Direct Sample MT511 Data - Block and Contract Level Data Flow Examples*
- *OASYS Global Direct Overview*
- *OASYS Global Direct Release Notes, Version 3.4.2*



Thomson Financial ESG

Thomson Financial ESG is the worldwide leader in providing technology-based workflow solutions to the global investment community. Thomson Financial ESG is committed to partnering with its clients to dramatically reduce the risk and cost of processing trades, and thereby improving their investment performance. Thomson Financial ESG services are used in 37 countries for domestic and cross-border trading. Thomson Financial ESG supports approximately 3,450 clients from 18 offices located in major financial centers across the globe.

Thomson Financial ESG is part of Thomson Financial (TF), a leading provider of information services and work solutions to the worldwide financial community. TF employs more than 7,000 people in more than 40 locations dedicated to the success of our clients and is part of The Thomson Corporation (TTC). With annual revenues approaching US$6 billion, TTC is one of the world's leading information and publishing companies. TTC's common shares are traded on the Toronto, Montreal and London stock exchanges

Thomson Financial ESG's products and services include ALERT®, ALERT*Direct*™, OASYS Global™, OASYS Global *Direct*™, OASYS™, OASYS *Direct*™, MarketMatch™, Thomson Report™, and AutoMatch™.

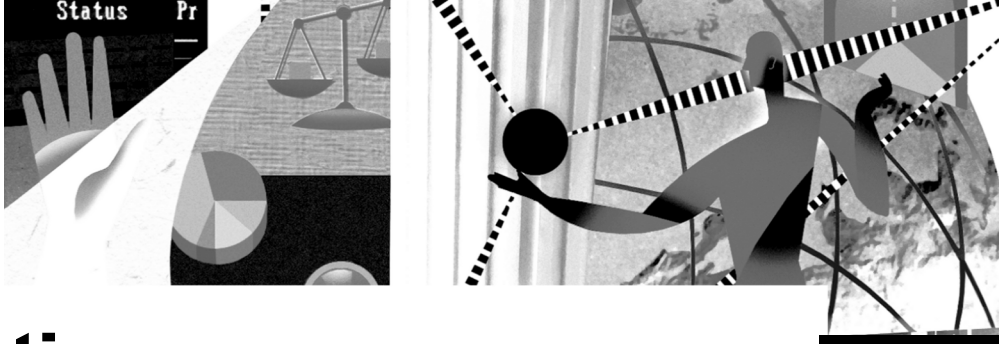
Sales Executives

Call your sales executives to learn more about other Thomson Financial ESG services that can automate your post-trade operations, including ALERT®, ALERT*Direct*™, AutoMatch™, OASYS Global™, OASYS Global *Direct*™, OASYS™, OASYS *Direct*™, MarketMatch™, and Thomson Report™. For assistance call the following sales offices.

Boston	1-800-407-4264	Paris	33-1-4453-7878
Chicago	1-312-629-0957	San Francisco	1-415-989-9797
Edinburgh	44-131-220-8417	Sao Paulo	5511-816-5022
Frankfurt	49-69-971-75200	Singapore	65-295-6383
Hong Kong	852-2905-3145	Stockholm	46-8-587-67160
London	44-207-369-7349	Sydney	612-9225-3158
Mexico City	525-535-6070	Thailand	001-800-65-1555
New York	1-212-612-9604	Tokyo	813-5218-6621

Electronic Trade Confirmation Code of Practice

See the "Electronic Trade Confirmation Code of Practice" section in *OASYS Global Direct Overview* for more information about the Electronic Trade Confirmation Code of Practice.



1: Introduction

The basic building block for exchanging trade confirmation information between trade parties (that is, brokers and investment managers) is the OASYS Global Message Type 511, or MT511. Modeled after the Securities Standards Advisory Board (SSAB) Message Type 511, the OASYS Global MT511 meets the trade confirmation needs of over 17 domestic markets with the flexibility to handle others. The MT511 is an ASCII, tag delimited, message format which Thomson Electronic Settlements Group (Thomson ESG) uses as the basis for the messages required for OASYS Global Electronic Trade Confirmation (ETC).

ETC sends and receives the MT511 based messages through the Message Delivery Service (MDS). The MDS is a guaranteed delivery, store, and forward system used to send and receive messages.

As OASYS Global MT511 messages travel back and forth, the OASYS Global host adds to them ISIN number and delivery instructions stored in the Thomson ESG ALERT database. These value added components are keyed off fields in the OASYS Global MT511 message.

This chapter provides a brief survey of the current OASYS Global system by showing the message flow and the MT511 message protocol use for the new Direct host-to-host connection. The rest of the document describes how to implement the required OASYS Global MT511 messages for timely electronic trade confirmation of securities.

How the System Works

The following diagram shows the various trade party connections to the Thomson ESG OASYS Global network.

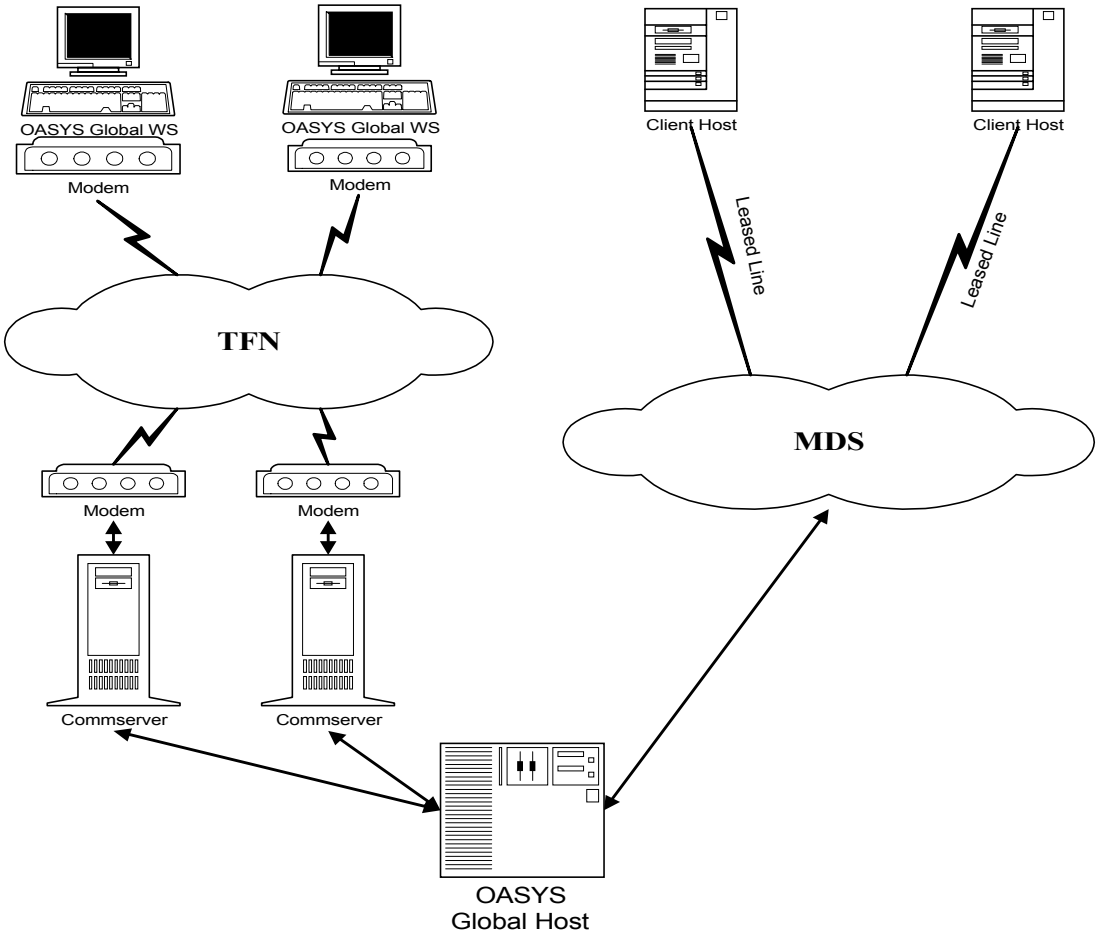
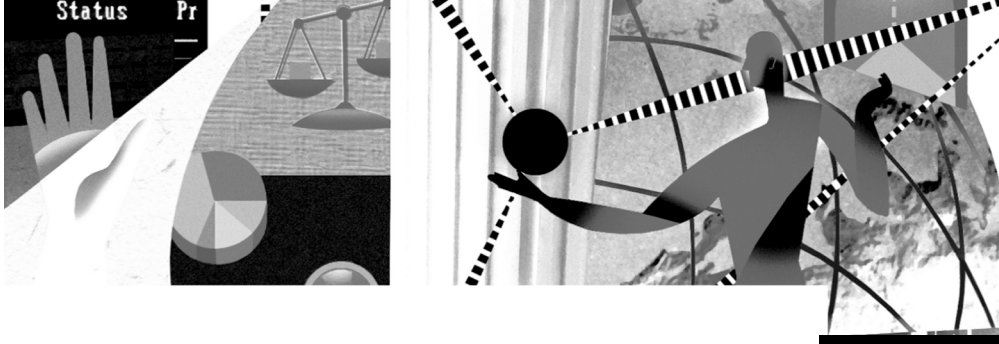


Figure 1.1

The left side of the diagram shows the existing OASYS Global system. Clients use OASYS Global workstations to enter data that then travels over modems and dial-in lines through the Thomson Financial Network (TFNNet) to communication servers at Thomson ESG. The OASYS Global host processes the incoming messages and forwards them by the communication servers to other OASYS Global workstations.

The right side of the diagram shows how to use the protocol, described in this document, for a new connection type, a host-to-host connection. Here, a client host computer formats trade messages in the OASYS Global MT511 format. These messages transit via leased lines through the Thomson ESG MDS to the OASYS Global host. From there, the host then routes them to other trade parties (who could be using OASYS Global workstations). Similarly, trade messages sent from an OASYS Global workstation pass through the host and the MDS to client hosts, where appropriate.



2: Definitions

This chapter defines terms related to OGD message processing. It contains the following sections:

MT511 Object

The MT511 object encapsulates all the information that the MT511 parser functions need to process received MT511 messages, to create new MT511 messages, and to handle any MT511 *Invalid* message and its associated reason (error) records.

MT511 Invalid Messages

When a host receives an MT511 message, it validates the message. If the validation fails, the host creates a new MT511 message of type *Invalid* and sends it to the originator of the message. An MT511 *Invalid* message contains two things:

1. The original MT511 message (that failed validation).
2. Reason codes and reason narratives explaining why the original MT511 message was invalid.

When the originator receives the MT511 *Invalid* message, it is responsible for determining the cause of the message and resolving the problem. A common resolution is to store the message (for example, in a database) for later examination by troubleshooting staff.

C Language Conventions

Throughout this document and in the sample code and test program, the following conventions apply:

`lower case`	Represents variables
`UPPERCASE`	Represents defined constants

Maximum Size for MT511 Messages

The largest MT511 message size is application dependent up to an absolute maximum. The defined constants `MT511_MAX_SIZE` and `MT511_REASON_REC_SIZE`, and the configuration parameter `MAX_MT511_ERRORS` define the buffer size for an MT511 message and an MT511 *Invalid* message.

The definitions of `MT511_MAX_SIZE` and `MT511_REASON_REC_SIZE` are in the file **mt511.h**. `MT511_MAX_SIZE` represents the maximum size of a normal MT511 message (anything but an MT511 *Invalid* message).

The value for the configuration parameter `MAX_MT511_ERRORS` must be within the range from 1 to 25 and defaults to 25. To change its value, call `MT511_config` with a new value within that range. This value defines the maximum size of an MT511 *Invalid* message according to the following formula, where `max_error_ct` represents the current value of the `MAX_MT511_ERRORS` parameter.

```
MT511_MAX_SIZE + (max_error_ct * MT511_REASON_REC_SIZE)
```

For messages received from the message switch (from Thomson ESG), the size of the receive buffer must be equal to the maximum size of an MT511 *Invalid* message. This maximum receive buffer size is defined by the following formula:

```
MT511_MAX_SIZE + (25 * MT511_REASON_REC_SIZE)
```

Repeating Groups/Fields

A repeating group is a set of associated MT511 fields that can occur multiple times in a single MT511 message. A repeating field is a repeating group with only one field. Refer to the *OASYS Global Direct MT511 Messaging Specification* for examples of repeating groups.

Multi-line Fields

Some MT511 fields, multi-line fields, can contain embedded newline characters. OASYS Global handles each multi-line field as a single string of the size of all lines, including the newline characters, although stripping of newlines may occur.

Examples of multi-line fields are as follows.

Tag	Description	Format	Field Length
:72C:	Reason Narrative	6*35x	215
:72B:	Further Information for the Party Identified	5*35x	179
:72:	Sender to Receiver Information	5*35x	179

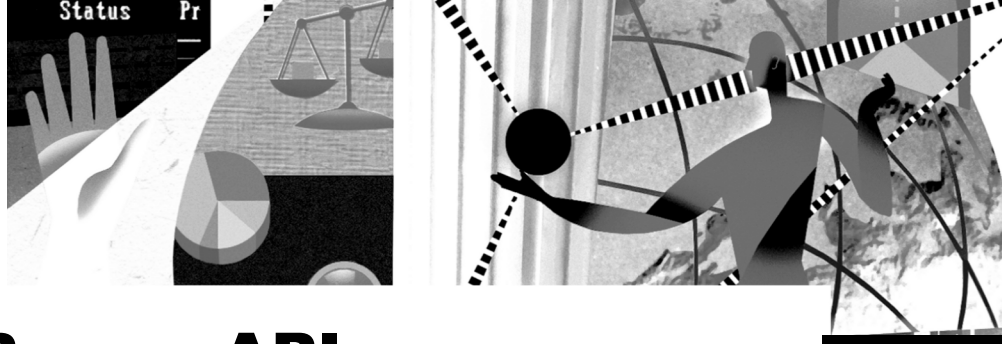
Tables

The parser can validate based on fixed sets of words located in user-defined tables. The ESG host's validation of incoming user messages was the intended use of this functionality, but it is also available to clients. Most clients validate table data before entering it to the parser.

The MT511 parser uses lookup tables and validation tables. The system uses a lookup table to return a value given a key. The lookup is only used for retrieving error text corresponding to an error number. The systems uses the data in a validation table to make sure that an MT511 message contains legal values. Tables may be hard-coded in flat files, be present in a database, or be stored in some other form.

The method your system uses to store tables determines how your system needs to open, start, and close them as flat files. Lookup and validation tables are part of the MT511 parser API in flat file form (see Appendix D, "MT511 Parser Installation Instructions," for the table names) to allow you to use the default callback functions (see below).

You may decide to validate or not validate the MT511 messages depending on the value set for a configuration parameter. If you decide to validate, then you must configure the validation tables.



3: MT511 Parser API Functions

This chapter provides detailed information about MT511 parser API functions. It contains the following sections:

The MT511 parser API consists of three types of functions: message functions, callback initialization functions, and default callback functions.

Message Functions

The MT511 parser API message functions are as follows.

Function	Description
`MT511_create`	Creates a new MT511 object instance.
`MT511_destroy`	Destroys an MT511 object instance.
`MT511_load_mt511`	Copies a received MT511 message into an MT511 object and parses and validates it.
`MT511_unload_mt511`	Obtains a copy of an MT511 message from an MT511 object.
`MT511_unload_invalid`	Obtains a copy of an MT511 *Invalid* message from an MT511 object.You, the client, do not normally use this function.
`MT511_add_reason`	Adds a reason (error) record to an MT511 object.
`MT511_get_field`	Obtains a field value from the MT511 message held by an MT511 object.
`MT511_get_first`	Obtains a field value from the first occurrence of a repeating group in the MT511 message held by an MT511 object.
`MT511_get_next`	Obtains a field value from the next occurrence of a repeating group in the MT511 message held by an MT511 object.
`MT511_put_field`	Copies a value into a field of the MT511 message held by an MT511 object.
`MT511_put_first`	Copies a value into the first occurrence of a repeating field of the MT511 message held by an MT511 object.
`MT511_put_next`	Copies a value into the next occurrence of a repeating field of the MT511 message held by an MT511 object.
`MT511_list_fields`	Writes to a file the field values and other internal data structures of the MT511 message held by an MT511 object.
`MT511_config_default`	Sets all the MT511 parser configuration parameters to their default values.
`MT511_config`	Sets MT511 parser configuration parameters.
`MT511GetVersion`	Returns the MT511 parser release version ID.

MT511_create

Prototype:

```
MT511* MT511_create()
```

Example:

```
MT511* mt511Obj;
mt511Obj = MT511_create();
```

Arguments:

None.

Return Values:

On success, the system returns a pointer to a structure of type MT511. If there is an error, the system returns the following value:

Return Value	Description
`NULL`	System did not create an MT511 object due to an unspecified error.

Description:

`MT511_create` allocates space for and initializes a new instance of an MT511 object.

Restrictions:

None.

Related Functions:

Most of the other MT511 parser API Message functions use the object returned by this function.

References:

The section titled “MT511 Object” in Chapter 2, “Definitions.”

MT511_destroy

Prototype:

```
int MT511_destroy(MT511* mt511Obj)
```

Example:

```
MT511* mt511Obj;
int status;
status = MT511_destroy(mt511Obj);
```

Arguments:

`MT511* mt511Obj` — A pointer to an MT511 object.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The system destroyed the object without error.
`FAILURE`	The system could not destroy the object due to an unspecified error.

Description:

`MT511_destroy` takes an existing MT511 object, deinitializes it, and releases the space allocated to it. Once your system destroys the object, you must not use the pointer to that object instance again.

Related Functions:

MT511_create

References:

None.

MT511_load_mt511

Prototype:

```
int MT511_load_mt511(MT511* mt511Obj, char* msgPtr)
```

Example:

```
MT511* mt511Obj;
char* msgPtr;
int status;
status = MT511_load_mt511(mt511Obj, msgPtr);
```

Arguments:

`MT511* mt511Obj` — A pointer to an MT511 object.

`char* msgPtr` — A pointer to an MT511 message.

Return Values:

The system returns the following values during normal operation. If the system encounters *n* syntax and validation errors during parsing and validation, it returns *n*. These are the fixed return values for this function:

Return Value	Description
`SUCCESS`	The message loaded, parsed, and validated without error.
`DETECTED_INVALID`	Your system determined the message was an MT511 *Invalid* message and therefore did not load it.
`FAILURE`	Your system could not load the message due to an unspecified error.

Description:

`MT511_load_mt511` copies an MT511 message into the MT511 object, then it parses and validates the message for syntactic and semantic correctness.

You can call an `MT511_load_mt511` only after a successful call to `MT511_create`. If `MT511_load_mt511` returns `SUCCESS`, your system can dispose of the original message buffer.

If `MT511_load_mt511` returns a value greater than `0`, indicating syntactic and/or semantic errors in the MT511 message, then it created and stored error messages, called reason records, in the MT511 object. Every time the system finds an error, it creates a reason record (up to a configurable maximum of 25, as specified by `MAX_MT511_ERRORS`). Reason records, along with the original MT511 message, contribute to an MT511 *Invalid* message when your system calls `MT511_unload_invalid`.

The host may find an MT511 message received from a client's site invalid and return it. If the returned message (or any such invalid message) passes to `MT511_load_mt511`, the system returns the value `DETECTED_INVALID`. The system does not store information about the message in the MT511 object, (that is, `MT511_unload_invalid` would fail if called). Since the MT511 parser does not save the message, your client application should save it to determine the nature of the problem.

The MT511 parser configuration parameter `LOAD_VALIDATE_DISABLED` (defaulted to `N`) controls whether the system semantically validates a message when loaded (not just syntactically verified).

3

Related Functions:

`MT511_unload`
`MT511_unload_invalid`
`log_msg` callback function

References:

Appendix A, “Validation Rules,” Appendix B, “MT511 Parser Configuration Parameters.”

MT511_unload_mt511

Prototype:

```
int MT511_unload_mt511(MT511* mt511Obj, char* bufPtr)
```

Example:

```
MT511* mt511Obj;
char buf[MT511_MAX_SIZE+1];
int status;
status = MT511_unload_mt511(mt511Obj, buf);
```

Arguments:

`MT511* mt511Obj` — A pointer to an MT511 object.

`char* bufPtr` — This is a buffer into which to copy the MT511 message. This buffer must be of size at least `MT511_MAX_SIZE` + 1.

Return Values:

The system returns the following values during normal operation. If the system encounters *n* syntax and validation errors during parsing and validation, it returns *n*; call `MT511_unload_invalid` to unload the message.

Return Value	Description
`SUCCESS`	A message was successfully created without syntactic or semantic errors.
`DETECTED_INVALID`	The message was determined to be an MT511 *Invalid* message and was not unloaded. Call `MT511_unload_invalid` instead.
`FAILURE`	A message could not be created due to an unspecified error.

Description:

`MT511_unload_mt511` creates a new MT511 message from the MT511 object and validates it for syntactic and semantic correctness. The system then copies the message into a supplied buffer.

If `MT511_unload_mt511` returns a value greater than `0`, indicating that it found syntactic and/or semantic errors in the MT511 message, then the system created error messages (reason records) and stored them in the MT511 object. The system creates a reason record for each error found (up to a configurable maximum of 25 as specified by `MAX_MT511_ERRORS`). The system uses the reason records, along with the original MT511 message, to construct an MT511 *Invalid* message when `MT511_unload_invalid` is called.

The MT511 parser configuration parameter `UNLOAD_VALIDATE_DISABLED` controls whether the system semantically validates a message when unloaded (rather than just syntactic verification). By default, the value of this parameter is `N`, which enables semantic validation.

Related Functions:

`MT511_unload_invalid`
`log_msg` callback function

References:

The section titled “Maximum Size for MT511 Messages” in Chapter 2, “Definitions.”
Appendix A, “Validation Rules,” Appendix B, “MT511 Parser Configuration Parameters.”

3

MT511_unload_invalid

Prototype:

```
int MT511_unload_invalid(MT511* mt511Obj, char* bufPtr)
```

Example:

```
MT511* mt511Obj;
char buf[MT511_MAX_SIZE + 25*MT511_REASON_REC_SIZE+1];
int status;
status = MT511_unload_invalid(mt511Obj, buf);
```

Arguments:

`MT511* mt511Obj` — A pointer to an MT511 object.

`char* bufPtr` — A buffer to copy the MT511 *Invalid* message into. This buffer must be of size at least `MT511_MAX_SIZE + (max_error_ct*MT511_REASON_REC_SIZE+1)` where `max_error_ct` represents the current value of the `MAX_MT511_ERRORS` configuration parameter; `max_error_ct` must be `25` if unloading a received MT511 *Invalid* message.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The system created an MT511 *Invalid* message without error.
`FAILURE`	The system could not create an MT511 *Invalid* message due to an unspecified error.

Description:

`MT511_unload_invalid` creates an MT511 *Invalid* message from the MT511 object incorporating any existing reason records and the original MT511 message. The system then copies the MT511 *Invalid* message into a supplied buffer.

Related Functions:

`MT511_unload_mt511`
`MT511_add_reason`
`log_msg` callback function

References:

The sections titled “MT511 Invalid Messages” and “Maximum Size for MT511 Messages” in Chapter 2, “Definitions.” Appendix B, “MT511 Parser Configuration Parameters.”

MT511_add_reason

Prototype:

```
int MT511_add_reason(MT511* mt511Obj, char* reasonCode,
char* subStr1, char* subStr2, char* subStr3, char* subStr4,
char* subStr5, char* subStr6)
```

Example:

```
MT511* mt511Obj;
int status;
status = MT511_add_reason(mt511Obj, "30000", "string 1",
"string 2", NULL);
```

Arguments:

`MT511* mt511Obj` — A pointer to an MT511 object.

`char* reasonCode` — A reason (error) code string, 16 characters or less (plus a 0-termination byte).

`char* subStri` — A pointer to one of six optional substitution strings, each 35 characters or less (plus a 0-termination byte). If fewer than six are supplied, pass `NULL` after the last.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`>0`	The system added a reason record without error.
`0`	The system did not add a reason record because a new record would exceed the maximum number configured.
`FAILURE`	The system could not add a reason record due to an unspecified error.

Description

`MT511_add_reason` builds a reason record out of a reason code and substitution strings, and stores it in the MT511 object along with any other existing reason records, in preparation for creating an MT511 *Invalid* message.

The system looks up the `reasonCode` string in the error code table (by default, **og_ecode.tab**). The result of this lookup is a sprintf-style format string with up to six "`%s`" format codes. The value of each `subStri` substitutes, in order, for each "`%s`" format. A `subStri` argument is required for each "`%s`" format. If fewer than six "`%s`" formats are provided for this `reasonCode` (i.e. fewer than six `subStri` arguments are passed) then `NULL` must be passed after the last `subStri` argument. Extra `subStri` arguments are ignored.

Note that `reasonCode 30000` is typically a special case. For this `reasonCode`, the `subStri` constitute a complete free-form text of the error message, and are catenated back to back (which means that an extra space may be needed at the beginning of one or more of the substitution strings for proper formatting). `NULL` may be passed if fewer than six `subStri` are provided.

The maximum number of reason records storable in an MT511 object is configurable up to 25. For more information, see the section titled "Maximum Size for MT511 Messages" in Chapter 2, "Definitions."

Related Functions:

MT511_unload_invalid

References:

The section titled “dflt_open_table” in this chapter. The sections titled “MT511 Invalid Messages” and “Maximum Size for MT511 Messages” in Chapter 2, “Definitions.” Appendix B, “MT511 Parser Configuration Parameters.”

MT511_get_field

Prototype:

```
int MT511_get_field(MT511* mt511Obj, int fieldId, char*
bufPtr)
```

Example:

```
MT511* mt511Obj;
char subMsgBuf[SUB_MSG_LEN + 1];
int status;
status = MT511_get_field(mt511Obj, FLD_SUB_MSG, subMsgBuf);
```

Arguments:

`MT511* mt511Obj` — A pointer to an MT511 object.

`int fieldId` — A defined constant (defined in **t_fldid.h**) identifying the MT511 field to get.

`char* bufPtr` — A buffer into which to copy the field value, which must be large enough to hold the largest field value for the desired field plus a 0-termination byte (the maximum width of each field, not including space for a 0-termination byte, is defined in **t_gbllen.h**).

Return Values:

The system returns the following values during normal operation:

Return Value	Description
l >0	The system retrieved the field value without error. *l* is the actual length of the field value not including the 0-termination byte.
0	The system did not retrieve the field value because the field was not present in the MT511 message.
FAILURE	The system did not retrieve the field value due to an unspecified error.

Description:

`MT511_get_field` copies a field value into a supplied buffer from the MT511 message held by an MT511 object or from the current group of an MT511 message held by an MT511 object.

You cannot call `MT511_get_field` before `MT511_load_mt511` has been successfully called. For fields in repeating groups, you cannot call `MT511_get_field` before successfully calling one or more of `MT511_get_first`, `MT511_get_next`, `MT511_put_first`, or `MT511_put_next`.

For repeating groups, `MT511_get_field` retrieves a field value from the current group. To traverse repeating groups (to change the current group), use functions `MT511_get_first`, `MT511_get_next`, `MT511_put_first`, and/or `MT511_put_next`.

Related Functions:

MT511_load_mt511
MT511_get_first
MT511_get_next
MT511_put_first
MT511_put_next

References:

The sections titled “Repeating Groups/Fields” and “Multi-line Fields” in Chapter 2, “Definitions.” Appendix C, “MT511 Tag to Parser Defined Constant Mapping,” and Appendix D, “MT511 Parser Installation Instructions.”

MT511_get_first

Prototype:

```
int MT511_get_first(MT511* mt511Obj, int fieldId, char*
bufPtr)
```

Example:

```
MT511* mt511Obj;
char partyTypeBuf[PARTY_TYPE_LEN + 1];
int status;
status = MT511_get_first(mt511Obj, FLD_TRADE_PARTY_TYPE,
partyTypeBuf);
```

Arguments:

`MT511* mt511Obj` — A pointer to an MT511 object.

`int fieldId` — A defined constant (defined in **t_fldid.h**) identifying the MT511 field to get.

`char* bufPtr` — A buffer into which to copy the field value, which must be large enough to hold the largest field value for the desired field plus a 0-termination byte (the maximum width of each field, not including space for a 0-termination byte, is defined in **t_gbllen.h**).

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`0`	The system did not retrieve the field value because the field was not present in the MT511 message.
l>`0`	The system retrieved the field value without error. *l* is the actual length of the field value not including the 0-termination byte.
`FAILURE`	The system did not retrieve the field value due to an unspecified error.

Description:

`MT511_get_first` copies a field value into a supplied buffer from the first occurrence of a repeating group in the MT511 message held by an MT511 object.

You cannot call `MT511_get_first` before successfully calling `MT511_load_mt511`.

For repeating groups, once you call `MT511_get_first`, the first repeating group becomes the current group. You can then call `MT511_get_field` and `MT511_put_field` to retrieve field values from and put field values into this current group, and you can call `MT511_get_next` and `MT511_put_next` to change the current group.

`MT511_get_first` returns `FAILURE` for an attempt to get field data from a grouped field that has no occurrences in an MT511 message rather than the expected `0` return value.

For retrieving values from fields not in repeating groups, use `MT511_get_field` instead.

Related Functions:

MT511_load_mt511
MT511_get_field
MT511_get_next
MT511_put_field

MT511_put_first
MT511_put_next

References:

The sections titled “Repeating Groups/Fields” and “Multi-line Fields” in Chapter 2, “Definitions.” Appendix C, “MT511 Tag to Parser Defined Constant Mapping,” and Appendix D, “MT511 Parser Installation Instructions.”

MT511_get_next

Prototype:

```
int MT511_get_next(MT511* mt511Obj, int fieldId, char*
bufPtr)
```

Example:

```
MT511* mt511Obj;
char partyTypeBuf[PARTY_TYPE_LEN + 1];
int status;
status = MT511_get_next(mt511Obj, FLD_TRADE_PARTY_TYPE,
partyTypeBuf);
```

Arguments:

`MT511* mt511Obj` — A pointer to an MT511 object.

`int fieldId` — A defined constant (defined in **t_fldid.h**) identifying the MT511 field to get.

`char* bufPtr` — A buffer into which to copy the field value, which must be large enough to hold the largest field value for the desired field plus a 0-termination byte (the maximum width of each field, not including space for a 0-termination byte, is defined in **t_gbllen.h**).

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`0`	The system did not retrieve the field value because the field was not present in the MT511 message.
l>`0`	The system retrieved the field value without error. *l* is the actual length of the field value not including the 0-termination byte.
`FAILURE`	The system did not retrieve the field value due to an unspecified error.

Description:

`MT511_get_next` copies a field value into a supplied buffer from the next occurrence of a repeating group in the MT511 message held by an MT511 object, in which the next group is relative to the current group. You cannot call `MT511_get_next` before successfully calling `MT511_get_first` or `MT511_put_first`.

For repeating groups, `MT511_get_next` changes the current group to be the next group relative to the current group. You can then call `MT511_get_field` and `MT511_put_field` to retrieve field values from and put field values into the current group, and you can call `MT511_get_next` and `MT511_put_next` again to change the current group.

If this function indicates that no next group is present in the message, then the current group does not change. For retrieving values from fields not in repeating groups, use `MT511_get_field` instead.

Related Functions:

MT511_load_mt511
MT511_get_field
MT511_get_first
MT511_put_field

MT511_put_first
MT511_put_next

References:

The sections titled “Repeating Groups/Fields” and “Multi-line Fields” in Chapter 2, “Definitions.” Appendix C, “MT511 Tag to Parser Defined Constant Mapping,” and Appendix D, “MT511 Parser Installation Instructions.”

MT511_put_field

Prototype:

```
int MT511_put_field(MT511* mt511Obj, int fieldId,
char* fieldValue)
```

Example:

```
MT511* mt511Obj;
int status;
status = MT511_put_field(mt511Obj, FLD_SUB_MSG, "SUB
MESSAGE");
```

Arguments:

`MT511* mt511Obj` — A pointer to an MT511 object.

`int fieldId` — A defined constant (defined in **t_fldid.h**) identifying the MT511 field to put.

`char* fieldValue` — A field value to put including a 0-termination byte. Excluding the 0-termination byte, this value must not be greater than the largest field value for the indicated field. The definition of maximum width of each field, not including space for a 0-termination byte, is in file **t_gbllen.h**.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The system copied the field value without failure.
`FAILURE`	The system could not copy the field value due to an unspecified error.

Descriptions:

`MT511_put_field` copies a field value into the MT511 message held by an MT511 object or into the current group of an MT511 message held by an MT511 object.

For fields in repeating groups, you cannot call `MT511_put_field` before you have successfully called one or more of `MT511_put_first`, `MT511_put_next`, `MT511_get_first`, or `MT511_get_next`.

After you call `MT511_put_field` and its companion functions `MT511_put_first` and `MT511_put_next` to add all the appropriate fields to the MT511 message, the calling application typically calls `MT511_unload_mt511`.

Related Functions:

MT511_unload_mt511
MT511_get_first
MT511_get_next
MT511_put_first
MT511_put_next

References:

The sections titled “Repeating Groups/Fields” and “Multi-line Fields” in Chapter 2, “Definitions.” Appendix C, “MT511 Tag to Parser Defined Constant Mapping,” and Appendix D, “MT511 Parser Installation Instructions.”

MT511_put_first

Prototype:

```
int MT511_put_first(MT511* mt511Obj, int fieldId,
char* fieldValue)
```

Example:

```
MT511* mt511Obj;
int status;
status = MT511_put_first(mt511Obj, FLD_TRADE_PARTY_TYPE,
"FIRST TYPE");
```

Arguments:

`MT511* mt511Obj` — A pointer to an MT511 object.

`int fieldId` — A defined constant (defined in **t_fldid.h**) identifying the MT511 field to put.

`char* fieldValue` — A field value to put including a 0-termination byte. Excluding the 0-termination byte, this value must not be greater than the largest field value for the indicated field. The definition of maximum width of each field, not including space for a 0-termination byte, is in file **t_gbllen.h**.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The system copied the field value without failure.
`FAILURE`	The system could not copy the field value due to an unspecified error.

Descriptions:

`MT511_put_first` copies a field value into the first occurrence of a repeating group in an MT511 message held by an MT511 object, creating this first group if necessary.

For repeating groups, once `MT511_put_first` is called, the first repeating group becomes the current group. You can then call `MT511_put_field` and `MT511_get_field` to put field values into and retrieve field values from the current group, and you can call `MT511_put_next` and `MT511_get_next` to change the current group.

For putting values into fields not in repeating groups, use `MT511_put_field` instead.

After you call `MT511_put_first` and its companion functions `MT511_put_field` and `MT511_put_next` to add all the appropriate fields to the MT511 message, the calling application typically calls `MT511_unload_mt511`.

Related Functions:

MT511_unload_mt511
MT511_get_field
MT511_get_first
MT511_get_next
MT511_put_field
MT511_put_next

References:

The sections titled “Repeating Groups/Fields” and “Multi-line Fields” in Chapter 2, “Definitions.” Appendix C, “MT511 Tag to Parser Defined Constant Mapping,” and Appendix D, “MT511 Parser Installation Instructions.”

MT511_put_next

Prototype:

```
int MT511_put_next(MT511* mt511Obj, int fieldId, char*
fieldValue)
```

Example:

```
MT511* mt511Obj;
int status;
status = MT511_put_next(mt511Obj, FLD_TRADE_PARTY_TYPE,
"NEXT TYPE");
```

Arguments:

`MT511* mt511Obj` — A pointer to an MT511 object.

`int fieldId` — A defined constant (defined in **t_fldid.h**) identifying the MT511 field to get.

`char* fieldValue` — A field value to put including a 0-termination byte. Excluding the 0-termination byte, this value must not be greater than the largest field value for the indicated field. The definition of maximum width of each field, not including space for a 0-termination byte, is in file **t_gbllen.h**.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The system copied the field value without failure.
`FAILURE`	The system could not copy the field value due to an unspecified error.

Descriptions:

`MT511_put_next` copies a field value into the next occurrence of a repeating group in an MT511 message held by an MT511 object, creating this next group if necessary, in which this next group is relative to the current group. You cannot call `MT511_put_next` before you have successfully called `MT511_put_first` or `MT511_get_first`.

For repeating groups, `MT511_put_next` changes the current group to be the next group relative to the current group. You can then call `MT511_put_field` and `MT511_get_field` to put field values into and retrieve field values from the current group, and you can call `MT511_put_next` and `MT511_get_next` again to change the current group.

For putting values into fields not in repeating groups, use `MT511_put_field` instead.

After `you call MT511_put_next` and its companion functions `MT511_put_field` and `MT511_put_first` to add all the appropriate fields to the MT511 message, the calling application typically calls `MT511_unload_mt511`.

Related Functions:

MT511_unload_mt511
MT511_get_field
MT511_get_first
MT511_get_next

MT511_put_field
MT511_put_first

References:

The sections titled “Repeating Groups/Fields” and “Multi-line Fields” in Chapter 2, “Definitions.” Appendix C, “MT511 Tag to Parser Defined Constant Mapping,” and Appendix D, “MT511 Parser Installation Instructions.”

MT511_list_fields

Prototype:

```
int MT511_list_fields(MT511* mt511Obj, char* filename)
```

Example:

```
MT511* mt511Obj;
int status;
status = MT511_list_fields(mt511Obj, "mt511.lis");
```

Arguments:

`MT511* mt511Obj` — A pointer to an MT511 object.

`char* filename` — A file into which to list the fields. If `NULL` is passed, the system lists the fields into **mt511.lis**.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The system wrote the field values without error.
`FAILURE`	The system did not write the field values due to an unspecified error.

Description:

`MT511_list_fields` writes to a file the field values of each field, as well as other internal data structures, in an MT511 message held by an MT511 object.

This function is provided as an aid to debugging.

Related Functions:

MT511_load_mt511
MT511_put_field
MT511_put_first
MT511_put_next

References:

Appendix C, "MT511 Tag to Parser Defined Constant Mapping."

MT511_config_default

Prototype:

```
int MT511_config_default()
```

Example:

```
int status;
status = MT511_config_default();
```

Arguments:

None.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
SUCCESS	The configuration parameters were set to their defaults without error.
FAILURE	The configuration parameters could not be set to their defaults due to an unspecified error.

Description:

`MT511_config_default` sets all of the MT511 parser configuration parameters to their default values.

You should call `MT511_config_default` and its companion function `MT511_config` during program initialization to set configuration parameter values before MT511 message processing begins.

Related Functions:

MT511_config

References:

Appendix B, “MT511 Parser Configuration Parameters.”

MT511_config

Prototype:

```
int MT511_config(int configParmId, char* parmValue)
```

Example:

```
int status;
status = MT511_config(MAX_MT511_ERRORS, "25");
```

Arguments:

`int configParmId` — defined constant identifying the configuration parameter to set. These constants are defined in **t_cfgid.h**.

`char* parmValue` — configuration parameter value to set, including a 0-termination byte.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The configuration parameter was set without error.
`FAILURE`	The configuration parameter could not be set due to an unspecified error.

Description:

`MT511_config` sets a specified MT511 parser configuration parameter to an indicated value.

You should call `MT511_config` and its companion function `MT511_config_default` during program initialization to set configuration parameter values before MT511 message processing begins.

Related Functions:

MT511_config_default

References:

Appendix B, "MT511 Parser Configuration Parameters."

MT511GetVersion

Prototype:

```
char* MT511GetVersion()
```

Example:

```
char* version;
version = MT511GetVersion();
```

Arguments:

None.

The system returns the following values during normal operation:

Return Value	Description
V	The calling application must not modify this value, the current MT511 parser release version ID.

Description:

`MT511GetVersion` returns the MT511 parser release version ID. This value tracks reported problems or errors with the MT511 parser. The client application should retrieve and report it at start-up time.

Related Functions:

None.

References:

None.

Callback Initialization Functions

The MT511 parser uses five callback functions. You must provide either the supplied default callback functions or a set of client application-defined functions. You set the callback functions using the callback initialization functions as follows.

Function	Description
`MT511_log_msg_fn`	Sets the function to call to report MT511 parser error and information messages.
`MT511_open_table_fn`	Sets the function to call to initiate access to a lookup or validation table.
`MT511_value_exists_fn`	Sets the function to call to look up a value in a validation table.
`MT511_close_table_fn`	Sets the function to call to terminate access to a lookup or validation table.
`MT511_get_error_text_fn`	Sets the function to call to retrieve from a lookup table an error format string associated with an error code.

Your application must set the callback functions during initialization by calling the appropriate Callback Initialization function. You must call to each of these functions. For example, the following illustrates initialization of the callback functions with the default callback functions:

```
MT511_log_msg_fn(&dflt_log_msg);
MT511_open_table_fn(&dflt_open_table);
MT511_value_exists_fn(&dflt_value_exists);
MT511_close_table_fn(&dflt_close_table);
MT511_get_error_text_fn(&dflt_get_error_text);
```

If you provide your own callback functions, the following restrictions apply:

1. You cannot use the names `log_msg`, `open_table`, `value_exists`, `close_table`, and `get_error_text`.
2. Your callback functions must present an interface identical to that of the default functions.

MT511_log_msg_fn

Prototype:

```
typedef int (*LogMsgFn)(int severity, int lineNo,
char* filename, char* function, char* format, ... )
int MT511_log_msg_fn(LogMsgFn funcPtr)
```

Example:

```
int status;
status = MT511_log_msg_fn(&dflt_log_msg);
```

Arguments:

`funcPtr` — Address of the callback function to call to report MT511 parser error and information messages. This function must present an interface identical to that of `dflt_log_msg`.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The system set the callback function without error.
`FAILURE`	The system could not set the callback function due to an unspecified error.

Description

`MT511_log_msg_fn` sets the function to call to report MT511 parser error and information messages.

Related Functions:

None.

References:

The section titled “dflt_log_msg” in this chapter.

MT511_open_table_fn

Prototype:

```
typedef int (*OpenTableFn)(char* tablename, int tableType,
int loadFlag)
int MT511_open_table_fn(OpenTableFn funcPtr)
```

Example:

```
int status;
status = MT511_open_table_fn(&dflt_open_table);
```

Arguments:

`funcPtr` — Address of the callback function to call to initiate access to a lookup or validation table. This function must present an interface identical to that of the `dflt_open_table` function described in this chapter.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The system set the callback function without error.
`FAILURE`	The system could not set the callback function due to an unspecified error.

Description

`MT511_open_table_fn` sets the function to call to initiate access to a lookup or validation table.

Related Functions:

MT511_value_exists_fn
MT511_close_table_fn
MT511_get_error_text_fn

References:

The sections titled “dflt_open_table” in this chapter and “Tables” in Chapter 2, “Definitions.”

MT511_value_exists_fn

Prototype:

```
typedef int (*ValueExistsFn)(int tableId, char* valueStr)
int MT511_value_exists_fn(ValueExistsFn funcPtr)
```

Example:

```
int status;
status = MT511_value_exists_fn(&dflt_value_exists);
```

Arguments:

`funcPtr` — Address of the callback function to call to call to look up a value in a validation table. This function must present an interface identical to that of the `dflt_value_exists` function described in this chapter.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The system set the callback function without error.
`FAILURE`	The system could not set the callback function due to an unspecified error.

Description

`MT511_value_exists_fn` sets the function to call to look up a value in a validation table.

Related Functions:

MT511_open_table_fn
MT511_close_table_fn
MT511_get_error_text_fn

References:

The sections titled “dflt_value_exists” in this chapter and “Tables” in Chapter 2, “Definitions.”

MT511_close_table_fn

Prototype:

```
typedef int (*CloseTableFn)(int tableId)
int MT511_close_table_fn(CloseTableFn funcPtr)
```

Example:

```
int status;
status = MT511_close_table_fn(&dflt_close_table);
```

Arguments:

`funcPtr` — Address of the callback function to call to terminate access to a lookup or validation table. This function must present an interface identical to that of the `dflt_close_table` function described in this chapter.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The system set the callback function without error.
`FAILURE`	The system could not set the callback function due to an unspecified error.

Description

`MT511_close_table_fn` sets the function to call to terminate access to a lookup or validation table.

Related Functions:

MT511_open_table_fn
MT511_value_exists_fn
MT511_get_error_text_fn

References:

The sections titled “dflt_close_table” in this chapter and “Tables” in Chapter 2, “Definitions.”

MT511_get_error_text_fn

Prototype:

```
typedef int (*GetErrorTextFn)(int tableId, char* errorCode,
char* formatBuf)
int MT511_get_error_text_fn(GetErrorTextFn funcPtr)
```

Example:

```
int status;
status = MT511_get_error_text_fn(&dflt_get_error_text);
```

Arguments:

`funcPtr` — Address of the callback function to call to retrieve, from a lookup table, an error format string associated with an error code. This function must present an interface identical to that of the `dflt_get_error_text` function described in this chapter.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The system set the callback function without error.
`FAILURE`	The system could not set the callback function due to an unspecified error.

Description

`MT511_get_error_text_fn` sets the function to call to retrieve, from a lookup table, an error format string associated with an error code.

Related Functions:

MT511_open_table_fn
MT511_value_exists_fn
MT511_close_table_fn

References:

The sections titled “dflt_get_error_text” in this chapter and “Tables” in Chapter 2, “Definitions.”

Default Callback Functions

The MT511 parser API includes the following default callback functions. You can pass these functions to the MT511 parser, via the Callback Initialization functions, if your client application does not define replacements. The MT511 parser will not execute unless you pass either the default functions or a set of client application-defined callback functions to the parser via the Callback Initialization functions.

These are the default callback functions:

Function	Description
`dflt_log_msg`	Writes a log message into a file.
`dflt_open_table`	Opens a lookup or validation table.
`dflt_value_exists`	Looks up a value in an open validation table.
`dflt_close_table`	Closes a lookup or validation table.
`dflt_get_error_text`	Retrieves, from a lookup table, an error format string associated with an error code.

dflt_log_msg

Prototype:

```
int dflt_log_msg(int severity, int lineNo, char* filename,
char* function, char* format, …)
```

Example:

```
int status;

status = dflt_log_msg(LOG_ERR, __LINE__, __FILE__,
"MT511_load_mt511",
Could not copy MT511buffer to MT511 object. Buffer is %s",
buffer);
```

Arguments:

`int severity` — A defined constant identifying one of the five severity levels for a message. These constants are defined in **mtcommon.h**.

`int lineNo` — The line number in the source file where the call to this function was made (e.g. the C predefined constant `_LINE_`).

`char* filename` — The name of the source file where the call to this function was made (e.g. the C predefined constant `_FILE_`).

`char* function` — The name of the routine where the call to this function was made.

`char* format` — A sprintf-style format string, which, in conjunction with the subsequent optional parameters, form a message to log.

… — Optional parameters formatted according to the format argument.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The system formatted the log message without error.
`FAILURE`	The system could not format the log message due to an unspecified error.

Description

`dflt_log_msg` formats a log message to the file **mt511err.out**. The severity codes and their descriptions are:

Severity Code	Description
`LOG_ERR`	A program error, not necessarily fatal.
`LOG_WARN`	An unusual non-error condition.
`LOG_MSG`	A milestone or general status information.
`LOG_DEBUG`	Verbose debugging information.
`LOG_ENTRY`	A function call trace.

Related Functions:

None.

3

References:

The section titled “MT511_log_msg_fn” in this chapter.

dflt_open_table

Prototype:

```
int dflt_open_table(char* tablename, int tableType, int
loadFlag)
```

Example:

```
int tableId;
tableId = dflt_open_table("broker.tab", TTYPE_BROK_ACRO,
FALSE);
```

Arguments:

`char* tablename` — The name of an existing table. When called by the MT511 parser, the value used for this parameter is passed to the parser during configuration.

`int tableType` — A defined constant identifying the type of table, which also determines its format. These constants are defined in **t_lkupid.h**.

`int loadFlag` — An advisory argument indicating whether this is a "quick response table" (`TRUE`) or a "standard response table" (`FALSE`). A "standard response table" should be cached in memory. This parameter is ignored by `dflt_open_table`.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
$t \geq 0$	The system opened the table with error. *t* represents the `tableId` passed to the companion functions `dflt_value_exists`, `dflt_close_table`, and `dflt_get_error_text`.
`FAILURE`	The system could not open the table due to an unspecified error.

Description

`dflt_open_table` opens a lookup or validation table of a given name and type.

The list of table types, their descriptions, their associated MT511 tags, and their standard file names is as follows. Unless otherwise indicated, all tables are validation tables.

Table Type	Description	MT511 Tag(s)	Standard Filename
`TTYPE_AGENCY`	A list of agency types.	`:83R:`	**agency.tab**
`TTYPE_ALERT_COUNTRY`	A list of ALERT country codes.	`:72B:/ALCC/`	**al_cntry.tab**
`TTYPE_ALERT_METHOD`	A list of ALERT clearing method codes.	`:72B:/ALMT/`	**clearing.tab**
`TTYPE_ALERT_SECTYPE`	A list of ALERT security type codes.	`:72B:/ALSC/`	**al_secty.tab**
`TTYPE_BOT_SOLD`	A list of MT511 bought/sold indicator codes.	`:23C:`	**og_buysl.tab**

`TTYPE_BROK_ACRO`	A list of broker acronyms.	`:80a1:`
		`:80a3:`
		`:80J:`
			broker.tab

`TTYPE_CHARGE_TYPE`	A list of charge type codes.	`:71B1:*1`	**og_chgty.tab**
`TTYPE_COMM_SHAR`	Unused.	N/A	N/A

Table Type	Description	MT511 Tag(s)	Standard Filename
TTYPE_CURR_CODE	A list of currency codes.	:33T2:*1 :34G:*2 :32M:*1 :23F:/ORG/*1 :34B:*1 :34H:*2 :33S:*1 :23F:/CLP/*1 :71B3:*1	**currency.tab**
TTYPE_ERR_CODES	A lookup table mapping error codes to error format strings. Used by dflt_get_error_text. See the section titled "dflt_get_error_text" in this chapter.	:TF06:	**og_ecode.tab**
TTYPE_INST_ACRO	A list of institution acronyms.	:80a1: :80a3: :80J:	**institut.tab**
TTYPE_PARTY	A list of MT511 party type codes.	:23K:	**party.tab**
TTYPE_QUAN_TYPE	A list of MT511 quantity type codes.	:35A:*1	**og_qtyty.tab**
TTYPE_RATING_TYPE	A list of MT511 rating type codes.	:23F:/RT/*1	**ratetype.tab**
TTYPE_REF_TYPE	A list of MT511 reference type codes.	:20A:	**og_refty.tab**
TTYPE_REPORT_DETAIL	A list of MT511 reporting detail type codes.	:23P:	**rep_dtl.tab**
TTYPE_SEC_CODES	A list of ALERT security type codes. A list of MT511 security codes.	:72B:/ALSC/ :35B:*1	**oa_sec.tab** **seccode.tab**
TTYPE_TRANS_COND	A list of transaction condition (bargain) codes.	:23J:	**bargain.tab**
TTYPE_YIELDS	A list of yield type codes.	:23F:/YLD/*1	**yield.tab**

Related Functions:

dflt_value_exists
dflt_close_table
dflt_get_error_text

References:

The sections titled "MT511_open_table_fn" in this chapter and "Tables" in Chapter 2, "Definitions." Appendix B, "MT511 Parser Configuration Parameters," and Appendix D, "MT511 Parser Installation Instructions."

dflt_value_exists

Prototype:

```
int dflt_value_exists(int tableId, char* value)
```

Example:

```
int tableId;
int status;
status = dflt_value_exists(tableId, "TSCO");
```

Arguments:

`int tableId` — ID of table in which to look up value. Returned from `dflt_open_table`.

`char* value` — Value to validate.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`cFOUND`	The value was found in the table.
`cNOTFOUND`	The value was not found in the table.
`FAILURE`	The lookup failed due to an unspecified error.

Description

`dflt_value_exists` looks up a value in a given open validation table. If this function always returns `cFOUND`, then only syntax checking is performed by the MT511 parser.

Related Functions:

dflt_open_table
dflt_close_table

References:

The sections titled "MT511_value_exists_fn" in this chapter and "Tables" in Chapter 2, "Definitions." Appendix B, "MT511 Parser Configuration Parameters," and Appendix D, "MT511 Parser Installation Instructions."

dflt_close_table

Prototype:

```
int dflt_close_table(int tableId)
```

Example:

```
int tableId;
int status;
status = dflt_close_table(tableId);
```

Arguments:

`int tableId` — ID of table to close. Returned from `dflt_open_table`.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`SUCCESS`	The table was closed without error.
`FAILURE`	The table could not be closed due to an unspecified error.

Description

`dflt_close_table` closes a given lookup or validation table.

Related Functions:

dflt_open_table
dflt_value_exists
dflt_get_error_text

References:

The sections titled “MT511_close_table_fn” in this chapter and “Tables” in Chapter 2, “Definitions.” Appendix B, “MT511 Parser Configuration Parameters,” and Appendix D, “MT511 Parser Installation Instructions.”

dflt_get_error_text

Prototype:

```
int dflt_get_error_text(int tableId, char* errorCode,
char* formatBufPtr)
```

Example:

```
int tableId;
char formatBuf[128];
int status:
status = dflt_get_error_text(tableId, "30020", formatBuf);
```

Arguments:

`int tableId` — ID of table from which to retrieve format string. Returned from `dflt_open_table`.

`char* errorCode` — Error code to look up.

`char* formatBufPtr` — A buffer to copy the format string into. This buffer must be at least 128 bytes in size.

Return Values:

The system returns the following values during normal operation:

Return Value	Description
`cFound`	The error code was found in the table and the format string copied into the format string buffer.
`cNotFound`	The error code was not found in the table.
`FAILURE`	The error code was not found in the table or the format string could not be copied into the format string buffer due to an unspecified error.

Description

`dflt_get_error_text` retrieves, from a given open lookup table, an error format string associated with a given error code.

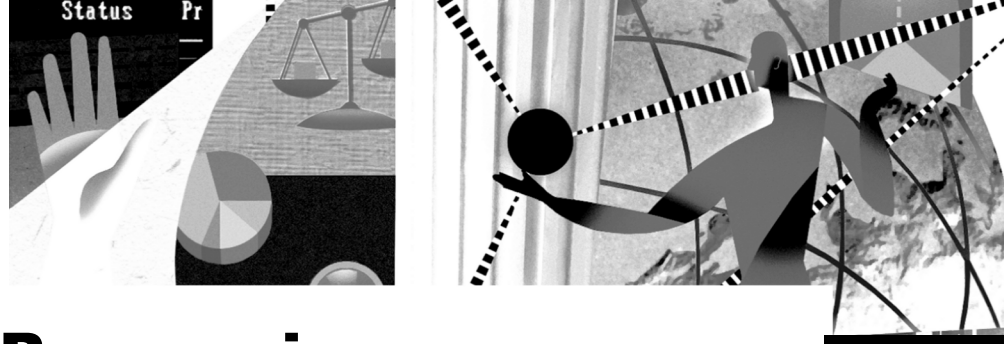
Related Functions:

dflt_open_table
dflt_close_table

References:

The sections titled "MT511_get_error_text_fn" in this chapter and "Tables" in Chapter 2, "Definitions." Appendix B, "MT511 Parser Configuration Parameters," and Appendix D, "MT511 Parser Installation Instructions."

3



4: Sample Processing

This chapter contains sample algorithms for processing received MT511 messages as well as creating new MT511 messages. Each section corresponds to a piece of sample code described in the following section and detailed in an external source file. This chapter contains the following sections:

Common Configuration

1. Call `MT511_log_msg_fn` to initialize the `log_msg` callback function.
2. Call `MT511_open_table_fn` to initialize the `open_table` callback function.
3. Call `MT511_value_exists_fn` to initialize the `value_exists` callback function.
4. Call `MT511_close_table_fn` to initialize the `close_table` callback function.
5. Call `MT511_get_error_text_fn` to initialize the `get_error_text` callback function.
6. Call `MT511_config_default`.
7. Call `MT511_config` to set any configuration parameters to values different from the defaults.

Receiving an MT511 Message

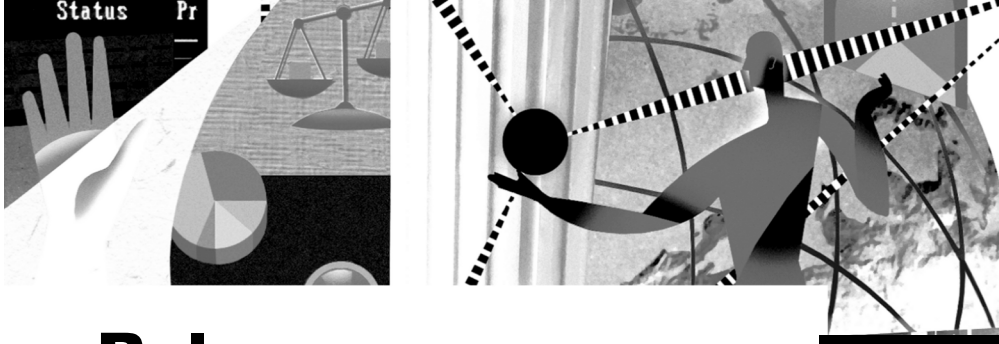
1. Receive the MT511 message from the messaging system and store it in a character buffer.
2. Call `MT511_create` to obtain a new instance of the MT511 object.
3. Call `MT511_load_mt511` to copy the stored MT511 message into the MT511 object.
   a. If `MT511_load_mt511` returns `SUCCESS`, access the fields of the MT511 message using calls to `MT511_get_field`, `MT511_get_first`, `MT511_get_next`.
   b. If `MT511_load_mt511` returns `DETECTED_INVALID` or an error count, store the MT511 message locally for debugging/problem resolution.
   c. If `MT511_load_mt511` returns `FAILURE`, Call your logging function to report the error. Call the Customer Service Center to report the error.
4. Call `MT511_destroy` when the application is done with the MT511 object.

Creating a New MT511 Message

1. Call `MT511_create` to obtain a new instance of the MT511 object.
2. Build a new MT511 message, field by field, by calling `MT511_put_field`, `MT511_put_first`, `MT511_put_next`.
3. Call `MT511_unload_mt511` to obtain a copy of the MT511 message buffer from the MT511 object.
4. Send the MT511 message, now in a character buffer, to the messaging system.
5. Call `MT511_destroy` when the application is done with the MT511 object.

When the application determines that an MT511 object is no longer needed, it should call `MT511_destroy` to free the object (and its allocated memory). It is important to free this memory as soon as possible to allow room for the next MT511 message and other allocated memory.

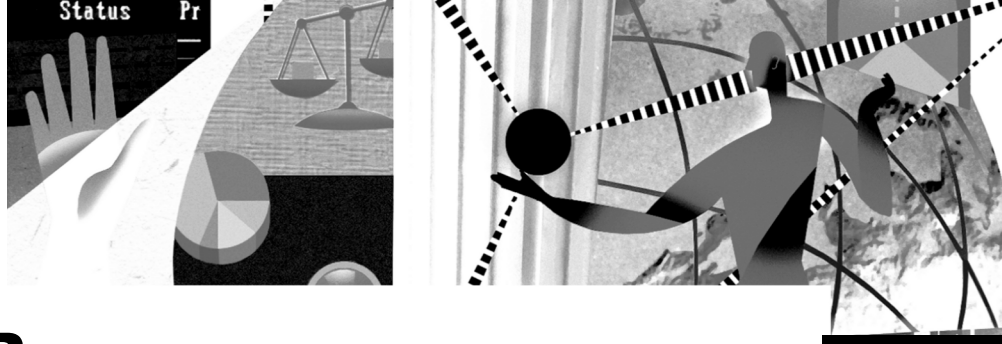
To display the internal data structures of the MT511 object (created by either `MT511_load_mt511` or the `MT511_put_*` functions), call `MT511_list_fields`.



A: Validation Rules

This appendix lists the rules that govern the process of MT511 message validation. The system creates reason records and reports errors if any piece of the MT511 message does not comply with these rules.

Rule	Description
Reasonable First Line in Message	Is the first line the protocol version or response code?
Mandatory Field Inclusion	Are all required fields for this message type included?
Field Allowed	Are all fields present allowed for this message type?
Repeating Group Counts	Do repeating groups have at least the minimum number of repeats and less than or equal to the maximum number?
Field Value Length	Are fixed-length field values the correct length? Are any values too long? Are any values too short?
Field Ranges	Are the field values within the ranges allowed?
Date and Time Checking	Are dates legal (1-12, 1-31, 1980-9999)? Are times legal (0-23, 0-59)?
Table Checks	Are fixed-choice items in the validation tables?



B: MT511 Parser Configuration Parameters

This appendix lists the parameters you use to configure the MT511 parser. Calling `MT511_config_default` sets these parameters to default values. Calling `MT511_config` sets these parameters to client-application values. The `configParmId` value documented for this function is one of those listed in the configuration parameter tables below.

For more information on table types (TTYPE_*), see the section titled "dflt_open_table" in Chapter 3, "MT511 Parser API Functions." For more information on log levels (LOG_*), see the section titled "dflt_log_msg" in Chapter 3, "MT511 Parser API Functions."

Table B-1 Configuration Parameters

configParmId	Description	Length[1]	Default Value
`ADD_TAG_COMMENTS`	Unused.	N/A	"N"
`AGENCY_TABLE`	Table name for type `ttype_agency`.	63	"" (no table)
`ALERT_COUNTRY_TABLE`	Table name for type `ttype_alert_country`.	63	"" (no table)
`ALERT_METHOD_TABLE`	Table name for type `ttype_alert_method`.	63	"" (no table)
`ALERT_SEC_TYPE_TABLE`	Table name for type `ttype_alert_sectype`.	63	"" (no table)
`ALL_MSG`	Unused.	N/A	""
`BOUGHT_SOLD_TABLE`	Table name for type `ttype_bot_sold`.	63	"" (no table)
`BROKER_TABLE`	Table name for type `ttype_brok_acro`.	63	"" (no table)
`CHARGE_TYPE_TABLE`	Table name for type `ttype_charge_type`.	63	"" (no table)
`COMM_SHARE_TYPE_TABLE`	Table name for type `ttype_comm_shar`.	63	"" (no table)
`COMMENTS_ALLOWED`	`Y` or `N`. If `Y`, then lines with the '#' symbol as the first character are ignored. Since multi-line fields may include a '#' symbol, they may be discarded. To avoid data loss, set this parameter to `N` in production.	1	"N"
`CURRENCY_TABLE`	Table name for type `ttype_curr_code`.	63	"" (no table)
`ENABLED_MSGS`	Message type the MT511 parser handles.	U[2]	"GBL"
`ERROR_CODE_TABLE`	Table name for type `ttype_err_codes`.	63	"" (no table)
`INSTITUTION_TABLE`	Table name for type `ttype_inst_acro`.	63	"" (no table)

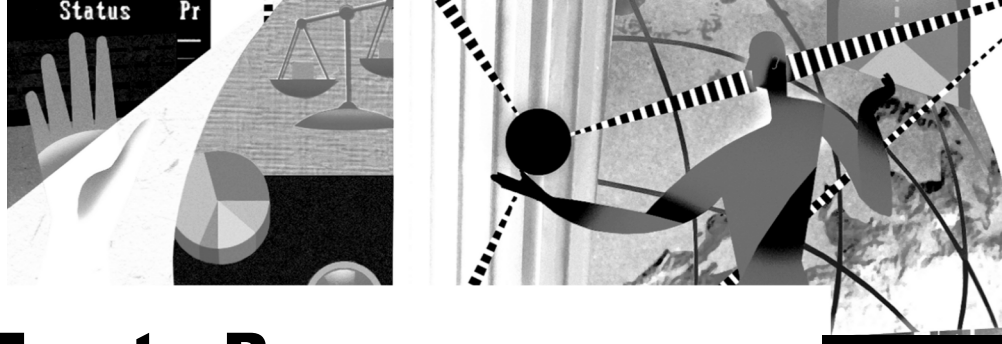
Table B-1 Configuration Parameters(Continued)

configParmId	Description	Length[1]	Default Value
LOAD_VALIDATE_DISABLE	Y or N. If Y, then skip validation during MT511_load_mt511.	1	"N"
MAX_MT511_ERRORS	0 to 25. Maximum number of reason records allowed before ending validation. If 0, use default limit. If 1 to 25, the error limit to set.	2	"10" "25"
NEWLINES_IN_FIELDS	Permits new lines in fields. Set this parameter to Y to process multi-line fields.	1	"N"
PARSER_TRACE_DEBUG	Y or N. If Y, then LOG_DEBUG-level messages are sent to the log_msg callback function. In production, set this to N.	1	"N"
PARSER_TRACE_ENTRY	Y or N. If Y, then LOG_ENTRY-level messages are sent to the log_msg callback function. In production, set this to N.	1	"N"
PARSER_TRACE_ERR	Y or N. If Y, then LOG_ERR-level messages are sent to the log_msg callback function. In production, set this to Y.	1	"Y"
PARSER_TRACE_MSG	Y or N. If Y, then LOG_MSG-level messages are sent to the log_msg callback function. In production, set this to Y.	1	"N"
PARSER_TRACE_WARN	Y or N. If Y, then LOG_WARN-level messages are sent to the log_msg callback function. In production, set this to Y.	1	"N"
PARTY_CODE_TABLE	Table name for type ttype_party.	63	"" (no table)
QTY_TYPE_TABLE	Table name for type ttype_quan_type.	63	"" (no table)
RATING_TYPE_TABLE	Table name for type ttype_rating_type.	63	"" (no table)
REF_TYPE_TABLE	Table name for type ttype_ref_type.	63	"" (no table)
REPORT_DETAIL_TABLE	Table name for type ttype_report_detail.	63	"" (no table)
SECURITY_CODE_TABLE	Table name for type ttype_sec_codes.	63	"" (no table)
TRANS_COND_TABLE	Table name for type ttype_trans_cond.	63	"" (no table)
UNLOAD_VALIDATE_DISABLED	Y or N. If Y, then skip validation during MT511_unload_mt511.	1	"N"
YIELD_CODE_TABLE	Table name for type ttype_yields.	63	"" (no table)

B

Notes: [1] Maximum length, excluding a 0-termination byte

[2] Undefined (controlled vocabulary)



C: MT511 Tag to Parser Defined Constant Mapping

Defined Constants

This appendix lists the parser MT511 field defined constants `fieldId` (i.e. those passed to the `MT511_get_field`, `MT511_get_first`, `MT511_get_next`, `MT511_put_field`, `MT511_put_first`, and `MT511_put_next` functions) as well as the field length defined constants associated with the MT511 tags.

Table C-1 Defined Constants

MT511 Tag	fieldId	Field Length
:12A:	FLD_SUB_MSG_VERSION	SUB_MSG_VERSION_LEN
:20A:	FLD_TRADE_PARTY_REF_TYPE	TRADE_PARTY_REF_TYPE_LEN
:20B:	FLD_TRADE_PARTY_REF	TRADE_PARTY_REF_LEN
:20C:	FLD_TFS_REF_VERSION	TFS_REF_VERSION_LEN
:23A:	FLD_SUB_MSG	SUB_MSG_LEN
:23B:	FLD_SUB_FUNCTION	SUB_FUNCTION_LEN
:23C:	FLD_BUY_SELL_IND	BUY_SELL_IND_LEN
:23F:/AMTX/	FLD_ALT_MIN_TAX_IND	ALT_MIN_TAX_IND_LEN
:23F:/BE/	FLD_BOOK_ENTRY_ONLY_IND	BOOK_ENTRY_ONLY_IND_LEN
:23F:/CFV/	FLD_CURRENT_FACE	CURRENT_FACE_LEN
:23F:/CLD/	FLD_CALL_DATE	CALL_DATE_LEN
:23F:/CLP/*1	FLD_CALL_PRICE_CODE	CALL_PRICE_CODE_LEN
:23F:/CLP/*2	FLD_CALL_PRICE	CALL_PRICE_LEN
:23F:/CLT/	FLD_CALL_TYPE	CALL_TYPE_LEN
:23F:/CPN/	FLD_COUPON_RATE	COUPON_RATE_LEN
:23F:/DD/	FLD_DATED_DATE	DATED_DATE_LEN
:23F:/FCT/	FLD_AMORTISED_FACTOR	AMORTISED_FACTOR_LEN
:23F:/FTX/	FLD_FED_TAX_IND	FED_TAX_IND_LEN
:23F:/ISR/	FLD_ISSUER	ISSUER_LEN
:23F:/MDD/	FLD_MATURITY_DATE	MATURITY_DATE_LEN
:23F:/OFCD/	FLD_ODD_FIRST_COUPON_DATE	ODD_FIRST_COUPON_DATE_LEN

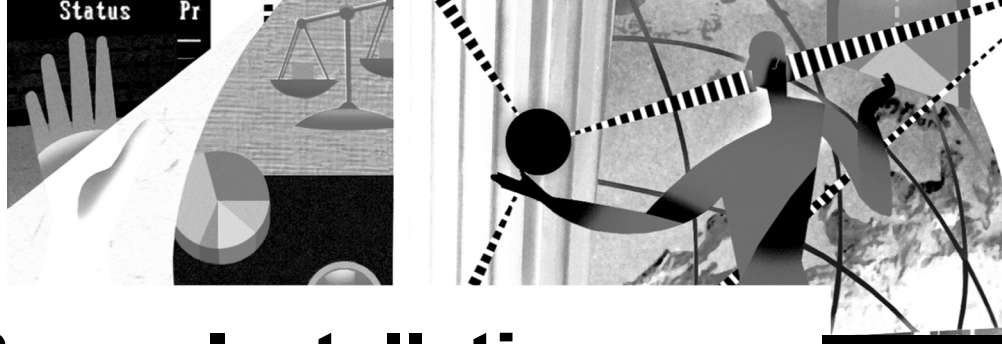
Table C-1 Defined Constants(Continued)

MT511 Tag	fieldId	Field Length
:23F:/ORG/*1	FLD_ORIG_FACE_AMT_CODE	ORIG_FACE_AMT_CODE_LEN
:23F:/ORG/*2	FLD_ORIG_FACE_AMT	ORIG_FACE_AMT_LEN
:23F:/RT/*1	FLD_RATING_TYPE	RATING_TYPE_LEN
:23F:/RT/*2	FLD_RATING_CODE	RATING_CODE_LEN
:23F:/YLD/*1	FLD_YIELD_TYPE	YIELD_TYPE_LEN
:23F:/YLD/*2	FLD_YIELD	YIELD_LEN
:23J:	FLD_TRANS_CONDITION	TRANS_CONDITION_LEN
:23K:	FLD_TRADE_PARTY_TYPE	TRADE_PARTY_TYPE_LEN
:23M:	FLD_RESPONSE_CODE	RESPONSE_CODE_LEN
:23O:	FLD_STAND_INSTRUCT_OV_IND	STAND_INSTRUCT_OV_IND_LEN
:23P:	FLD_REPORTING_DETAIL	REPORTING_DETAIL_LEN
:23Q:	FLD_COMM_SHARING_TYPE	COMM_SHARING_TYPE_LEN
:30S2:*1	FLD_ALLOC_SETL_DATE	ALLOC_SETL_DATE_LEN
:31P2:	FLD_DATE_OF_TRADE	DATE_OF_TRADE_LEN
:31P4:	FLD_MARKET	MARKET_LEN
:31T:	FLD_TIME_OF_TRADE	TIME_OF_TRADE_LEN
:32M:*1	FLD_DEAL_AMT_CODE	DEAL_AMT_CODE_LEN
:32M:*2	FLD_DEAL_AMT	DEAL_AMT_LEN
:33S:*1	FLD_SPECIAL_CONCESSIONS_CODE	SPECIAL_CONCESSIONS_CODE_LEN
:33S:*2	FLD_SPECIAL_CONCESSIONS	SPECIAL_CONCESSIONS_LEN
:33S:*3	FLD_SPECIAL_CONCESSIONS_EXT	SPECIAL_CONCESSIONS_EXT_LEN
:33T2:*1	FLD_TRANS_PRICE_CODE	TRANS_PRICE_CODE_LEN
:33T2:*2	FLD_TRANS_PRICE	TRANS_PRICE_LEN
:33T4:	FLD_AVG_PRICE_IND	AVG_PRICE_IND_LEN
:34B:*1	FLD_NET_PROCEEDS_CODE	NET_PROCEEDS_CODE_LEN
:34B:*2	FLD_NET_PROCEEDS	NET_PROCEEDS_LEN
:34G:*1	FLD_ACCRUED_INT_DAYS	ACCRUED_INT_DAYS_LEN
:34G:*2	FLD_ACCRUED_INT_CODE	ACCRUED_INT_CODE_LEN
:34G:*3	FLD_ACCRUED_INT	ACCRUED_INT_LEN
:34H:*1	FLD_ACCRUED_INT_NEG_DAYS	ACCRUED_INT_NEG_DAYS_LEN
:34H:*2	FLD_ACCRUED_INT_NEG_CODE	ACCRUED_INT_NEG_CODE_LEN
:34H:*3	FLD_ACCRUED_INT_NEG	ACCRUED_INT_NEG_LEN
:35A:*1	FLD_QTY_CODE	QTY_CODE_LEN
:35A:*2	FLD_QTY	QTY_LEN
:35B:*1	FLD_ID_FNCL_INSTR_CODE_TYPE	ID_FNCL_INSTR_CODE_TYPE_LEN
:35B:*2	FLD_ID_FNCL_INSTR_CODE	ID_FNCL_INSTR_CODE_LEN
:35B:*3	FLD_ID_FNCL_INSTR_DESCR	ID_FNCL_INSTR_DESCR_LEN

C

Table C-1 Defined Constants(Continued)

MT511 Tag	fieldId	Field Length
:36I:	FLD_DEAL_AMT_XCHG_RATE	DEAL_AMT_XCHG_RATE_LEN
:71B1:*1	FLD_CHARGE_TAX_TYPE	CHARGE_TAX_TYPE_LEN
:71B1:*2	FLD_CHARGE_TAX_TYPE_EXT	CHARGE_TAX_TYPE_EXT_LEN
:71B3:*1	FLD_CHARGE_AMOUNT_CODE	CHARGE_AMOUNT_CODE_LEN
:71B3:*2	FLD_CHARGE_AMOUNT	CHARGE_AMOUNT_LEN
:72:	FLD_SENDER_TO_RCVR_INFO	SENDER_TO_RCVR_INFO_LEN
:72A:	FLD_RESPONSE_NARRATIVE	RESPONSE_NARRATIVE_LEN
:72B: /ACCTREF/	FLD_TRADE_PARTY_ACCT_REF	TRADE_PARTY_ACCT_REF_LEN
:72B:/ALAC/	FLD_ALLOC_ALERT_ACCESS_CODE	ALLOC_ALERT_ACCESS_CODE_LEN
:72B:/ALCC/	FLD_ALLOC_ALERT_COUNTRY_CODE	ALLOC_ALERT_COUNTRY_CODE_LEN
:72B:/ALDN/	FLD_ALERT_DELIVERY_NAME	ALERT_DELIVERY_NAME_LEN
:72B:/ALMT/	FLD_ALERT_METHOD_TYPE	ALERT_METHOD_TYPE_LEN
:72B:/ALSC/	FLD_ALERT_SECURITY_TYPE	ALERT_SECURITY_TYPE_LEN
:72B:/NAM/	FLD_PARTY_NAME	PARTY_NAME_LEN
:72C:	FLD_REASON_NARRATIVE	REASON_NARRATIVE_LEN
:80J:	FLD_PARTY_ID	PARTY_ID_LEN
:80a1:	FLD_SENDER_ACRO	SENDER_ACRO_LEN
:80a3:	FLD_RCVR_ACRO	RCVR_ACRO_LEN
:83R:	FLD_AGENCY_PRINC_IND	AGENCY_PRINC_IND_LEN
:TF01:	FLD_ADVICE_IND	ADVICE_IND_LEN
:TF06:	FLD_REASON_CODE	REASON_CODE_LEN (none defined)
:TF09:/PCT/	FLD_SPECIAL_CONCESSIONS_STRING_PCT	SPECIAL_CONCESSIONS_STRING_PCT_LEN
:TF09:/RAT/	FLD_SPECIAL_CONCESSIONS_STRING_RAT	SPECIAL_CONCESSIONS_STRING_RAT_LEN
:TF10:	FLD_SETTLEMENT_CODE	SETTLEMENT_CODE_LEN
:TF11:	FLD_SETTLEMENT_VALUE	SETTLEMENT_VALUE_LEN
:TF12:	FLD_DATE_OF_MSG	DATE_OF_MSG_LEN
:TF13:	FLD_TIME_OF_MSG	TIME_OF_MSG_LEN
:TF14:	FLD_PARTY_REF	PARTY_REF_LEN
:TF15:	FLD_LOT_SIZE	LOT_SIZE_LEN
:TFH1:	FLD_MSG_TYPE	MSG_TYPE_LEN
:TFH2:	FLD_FIRST_SUB_MSG	FIRST_SUB_MSG_LEN
:TFH3:	FLD_LAST_SUB_MSG	LAST_SUB_MSG_LEN
:TFH9:	FLD_PROTOCOL_VERSION	PROTOCOL_VERSION_LEN



D: MT511 Parser Installation Instructions

This appendix provides MT511 parser installation information. It contains the following sections:

The OASYS Global MT511 parser distribution kit includes the include files, parser library, default tables, default callback functions, and sample code files.

MOA Communications Upgrade

OGD has been upgraded to version 6.8 of the communications Messaging Open Application Programmers' Interface (MOA API). These changes are required to existing applications:

- For Windows NT: The MOA for Windows NT uses the **wsock32.dll** Windows socket library to communicate with the host via TCP/IP. A new **moa.lib** library works with the **winmoa32.lib** library from OGD version 3.3.

> ***Note!*** There is a new MOA_SET program. You must regenerate your MDS configuration file using the new MOA_SET program after rebuilding. See Chapter 3, "MOA Configuration," in the Message Delivery System TCP/IP API Programmer's Guide for more information.

- For Solaris: For the Sun, the shared library **libMOA.so** and its static counterpart **libMOA.a** have replaced the **libmds.a** statically linked library. These files must be put in a known common place (for example, `/usr/local/lib`). To link, pass the flags `-L/usr/local/lib -lMOA` (substituting the appropriate directory for the `-L` flag). To run using this shared library, add `/usr/local/lib`, or the appropriate directory, to the `LD_LIBRARY_PATH` environment variable of the UNIX user that starts the client application. Note that if you actually use `/usr/local/lib`, you probably will not need to modify `LD_LIBRARY_PATH`.

D

Include Files

The following include files are distributed with the MT511 parser.

Client-Application Include Files

Use the `#include` statement to include the following file into the client application code:

mt511g.h

Other Include Files

The following files are distributed with the MT511 parser but you should not directly include them into the client application code:

dflt_tbl.h
fldstore.h
general.h
log.h
mt511.h
mtcommon.h
mtconfig.h
mtgetput.h
mtload.h
mtunload.h
stdinc.h
t_cfgid.h
t_errid.h
t_fldid.h
t_gbllen.h
vector.h

Parser Library

The following file is linked with the client's application code:

ogparser.a

Default Tables

The following tables, which are used by the default callback functions, are distributed with the MT511 parser. For more information, see the sections titled "Callback Initialization Functions" and "Default Callback Functions" in Chapter 3, "MT511 Parser API Functions"

Table Name	Table Description
agency.tab	Agency/Principal/Cross Trade Indicator table. OGD uses the codes in this table in field 22, tag 83R. See the description of field 22 under "Field Tag 83R: Agency/Principal/Cross Trade Indicator" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
al_cntry.tab	ALERT Country Code table. OGD uses the codes in this table in field 46, tag 72B/ALCC/. See the description of field 46 under "Field Tag 72B: Further Information for Party Identified" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
al_secty.tab	ALERT Security Type Code table. OGD uses the codes in this table in field 46, field tag 72B/ALSC/. See the description of field 46 under "Field Tag 72B: Further Information for Party Identified" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
al_xref.tab	OG Broker/Institution to ALERT Broker/Institution Mapping table. Note that only ALERT brokers and institutions can access the ALERT database.
bargain.tab	Transaction Condition Code table. OGD uses the codes in this table in field 41, field tag 23J. See the description of field 41 under "Field Tag 23J: Transaction Condition" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
broker.tab	Broker Code table. List of all broker codes.
clearing.tab	ALERT Clearing Method Code table. OGD uses the codes in this table in field 46, field tag 72B / ALMT/. See the description of field 46 under "Field Tag 72B: Further Information for Party Identified" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
currency.tab	Currency Code table. Currency codes appear in the 3a segment of all fields used for monetary values. Precision determines limits of certain numeric fields. See "Field Tag 36I: Exchange Rate"" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document.for specific field detail.
institut.tab	Institution Code table. List of all institution codes.
og_buysl.tab	Bought/Sold Indicator table. OGD uses the codes in this table in field 21, field tag 23C. See the description of field 21 under "Field Tag 23C: Bought/Sold Indicator" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
og_chgty.tab	Charge/Tax Type table. OGD uses the codes in this table in field 61, field tag 71B1. See the description of field 61 under "Field Tag 71B1: Charge/Tax Type" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
og_ecode.tab	Reason Code table. OGD uses the codes in this table in field 17, field tag TF06. See the description of field 17 under "Field Tag TF06: Reason Code" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.

Table Name	Table Description
og_party.tab	Party Type Code table. OGD uses the codes in this table in field 43, field tag 23K. See the description of field 43 under "Field Tag 23K: Party Type" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
og_qtyty.tab	Quantity of Financial Instrument table. OGD uses the codes in this table in field 23, field tag 35A. See the description of field 23 under "Field Tag 35A: Quantity of Financial Instrument" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
og_ratet.tab	Rating Type table. OGD uses the codes in this table in field 40, field tag 23F/RT/. See the description of field 40 under "Field Tag 23F: Financial Instrument Attribute" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
og_refty.tab	Type of Sub-Message Reference table. OGD uses the codes in this table in field 12, field tag 20A. See the description of field 12 under "Field Tag 20A: Type of Sub-Message Reference" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
og_repdt.tab	Reporting Detail table. OGD uses the codes in this table in field 81, field tag 23P. See the description of field 81 under "Field Tag 23P: Reporting Detail" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
seccode.tab	Security Identifier Types Code table. This is a list of security identifiers that describe the type of security identification used on a message. These codes are valid for field 39, field tag 35B. See the description of field 39 under "Field Tag 35B: Identification of Financial Instrument" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.
og_yield.tab	Yield table. OGD uses the codes in this table in field 40, field tag 23F/YLD/. See the description of field 40 under "Field Tag 23F: Financial Instrument Attribute" in Appendix A, "MT511 Data Dictionary," in the *OASYS Global Direct MT511 Messaging Specification* document for more information.

D

Default Callback Functions

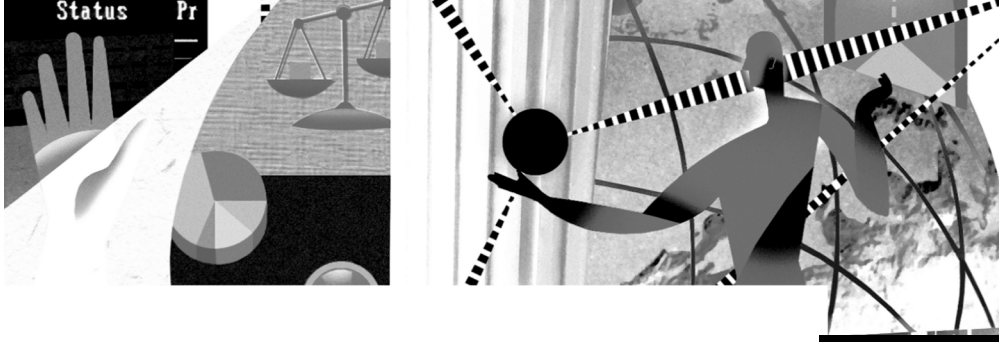
The source code for the default callback functions can be found in the following file:

dflt_tbl.c

Sample Code

The following files contain sample code demonstrating how to use the MT511 parser:

receive.c
send.c
makefile



Index

A

B

C

D

E

F

G

I

L

M

O

P

R

S

T

V

W

Y