StoreNext

ISS45 User Hooks

for ISS45 V7 and V8

ISS45 V7 and V8 User Hooks

Date of Issue	Product ID Num.	Part Number	Brief Description
March 2010	45001/116	89000561	Initial Release

Address comments and corrections to:

Software Program Director
StoreNext Retail Technologies LLC
6100 Tennyson Parkway / Suite 130 / Plano, Texas 75024

Table of Contents

Overview

From time to time there may be the need for a grocer to perform additional functionality outside of the POS application. ISS45 offers the ability to call additional programs before, during and after a transaction. This functionality is achieved by the use of Custom User Hooks. These Custom Hooks are user-definable applications that can be activated before or after certain functions at the POS terminals.

Some of the benefits that can be achieved by the use of Custom Hooks are:

- Support of specific business rules.
- Support applications outside of ISS45 during a transaction to preserve legacy interface file formats.
- The ability to implement a temporary work around while waiting for the next ISS45 release.
- As an additional alternative to providing custom software at the POS.
- To reduce the amount of development required from StoreNext.

Examples of task that can be performed by using Custom Hooks are:

- Modify the look of the sales receipt.
- Prompt the cashier for additional information.
- Inject items into a transaction.
- Capture information during a transaction and write the information to other files/systems.
- Provide cashier functionality not offered in the base ISS45 system.

Having this functionality available in ISS45 gives grocers the flexibility to create custom applications that can extend the functionality of ISS45.

Note:

Custom Hooks can also be performed using CDS (Code Distribution Services). The CDS engine provides user hooks to extend the standard functionality of the software upgrade process. These user hooks allow the user to deploy customized files and perform additional system maintenance at the conclusion of the standard CDS process. CDS Hooks can also be applied at the POS terminals even if there is no need to run an ISS45 upgrade. For more information, please see the **CDS Administrator's Guide**.

POS Hooks

POS Hooks are linked into the POS application via DLLs. The WinPOS application by default has “empty” DLLs for each Hook. To utilize a Hook at the POS, replace the empty DLL with a new DLL that contains the code for the additional functionality. The Hook DLLs are written in C, C++. Furthermore, they can be wrappers for functions written in Java, .NET, etc.

There is one source file that you can update, according to your requirements.

File	Description / Use
WPHOOK.C	Hook functions are self-explanatory. Please note that you must activate the hooks in the system (see below) as well as the specific hook that is used.

POST User Hook Parameters

The following ISS45 hook parameters are located in:

- General System Parameters (under the Store \ POST \ Operational \ Post User Hooks folder) for ISS45 v8.
- The POST User Hooks (Page 1 through Page 3) Screens (6-1-3-4) in ISS45 v7.

The parameters specify whether the POS Terminal can activate this hook.

User Hooks in System: This parameter specifies that one or more User Hooks are enabled at the POST.

Debug Mode While in User Hooks: This parameter provides the option of entering Debug Mode when User Hooks are enabled.

Initialization Hook: This parameter specifies that this Hook will be activated during the Initialization of the POST.

Before No-Sale Operation: This parameter specifies that the Hook is activated just before the start of the next transaction, i.e., after the drawer is closed following the previous operation, for example, Sale Transaction, Pickup, Sign-on, etc.

At End of Transaction: This parameter specifies that the Hook is activated at the end of each Sales Transaction, before printing the Trailer message and before sending the Transaction to the Host.

Before Sign-On, Sign-Off, Secure or Wait Mode: This parameter specifies that the Hook is executed before Sign-on, Sign-off, Secure mode and Start/Start Wait mode.

After Total Key is Pressed: This parameter specifies that the Hook is activated immediately after the Subtotal key is pressed but before any operation is executed.

After Total Key and Order Promotion Calculation: This parameter specifies that the Hook is activated after pressing the Total key and the balance due is adjusted according to Order Promotion.

Before PLU Read: This parameter specifies that the Hook is executed after entering or scanning an item but before the item information is read from the item file.

After PLU Read: This parameter specifies that the Hook is executed after the item information is read but before the information is processed in any way.

After PLU Sale Completed: This parameter specifies that the Hook is executed after the Item Transaction is completed.

Before Department Read: This parameter specifies that the Hook is executed after entering a department but before the department information is read from the Item file. The return from the Hook will determine if the Department Sale execution is continued or terminated.

After Department Read: This parameter specifies that the Hook is executed after the item information is read but before the information is processed in any way. The return from the Hook will determine if the Department Sale execution is continued or terminated.

After Department Completion: This parameter specifies that the Hook is executed after the Department Sale is completed.

Before Discount Read: This parameter specifies that the Hook is executed after selecting the discount but before the discount information is read from the discount file. The return from the Hook will determine if the Discount execution is continued or terminated.

After Discount Read: This parameter specifies that the Hook is executed after the discount information is read but before the information is processed in any way. The return from the Hook will determine if the Discount execution is continued or terminated.

After Discount Completion: This parameter specifies that the Hook is executed after the Discount is completed.

Before Return Read Hook: This parameter specifies that the Hook is executed after selecting the return but before the return information is read from the returned file. The return from the Hook will determine if the Return execution is continued or terminated.

After Return Read Hook: This parameter specifies that the Hook is executed after the return information is read but before the information is processed in any way. The return from the Hook will determine if the Return execution is continued or terminated.

Before Tender Read Hook: This parameter specifies that the Hook is executed after selecting the tender but before the tender information is read from the Tender file. The return from the Hook will determine if the Tender execution is continued or terminated.

After Tender Read Hook: This parameter specifies that the Hook is executed after the Tender information is read but before the information is processed in any way. The return

from the Hook will determine if the Tender execution is continued or terminated.

After Tender Completion Hook: This parameter specifies that the Hook is executed after tendering is completed.

After Frequent Shopper Entry Hook: This parameter specifies that the Hook is executed after the Frequent Shopper entry but before it is validated. The return from the Hook will determine whether to continue or re-enter the Frequent Shopper entry.

Before "Check Signature" Prompt Hook: This parameter specifies that the Hook is executed just before the Check Signature prompt. The return from the Hook will determine whether to continue or terminate the Tender execution.

User Hook Functions Hook: This parameter enables the Hook that executes the various User Hook functions. The Hook function must first be added into the keyboard layout or into the POST menu. Using a Hook function will result in execution of this Hook with the Hook function number.

Before BROM Read: This parameter specifies that the Hook is executed just before promotion information is read from the database.

After BROM Read: This parameter specifies that the Hook is executed just after promotion information is read from the database.

After BROM Reduction Completion: This parameter specifies that the Hook is executed after Reduction is completed. Only Reductions have an after Completion Hook because Promotions and Offers are executed as a special PLU sale, therefore, the PLU After Completion Hook covers this purpose.

Before Maintenance Execution: This parameter specifies that the Hook is executed just before every maintenance execution. The return from the Hook will determine whether to continue execution of this maintenance (OK/BAD).

After Maintenance Execution: This parameter specifies that the Hook is executed just after every maintenance execution.

Implementation Example

The function “ppm_exe_print” is supported to print information on the receipt through a hook function. If a retailer wants to print additional information on the receipt, the hook function “hook_at_end_of_transaction” can be used.

To print information on the receipt, the POSWare Presentation Manager can be used to create a POS Format (“**CUSTINFO**” for example) using the “ppm_exe_print” function to print information through a hook function.

The POS Format used would be as follows:

Line 1: Format name, label name (or use NULL if no label name)

Line 2: Output name defined in POS format. (Use FLD_TYPE_LONG if a long type value is printed)

Line 3: Output name defined in POS format (Use FLD_TYPE_ALPHA if a char type value is printed)

Line 4: Retailer can define additional output names in the POS Format to print multiple line information.

Line 5: Always use NULL to be a last parameter when calling ppm_exe_print function.

Example

```
Line1-  ppm_exe_print (“CUSTINFO", NULL,
Line2-          "variableLong",  FLD_TYPE_LONG,  0,  (long)SaleAmountInfo,
Line3-          "variableChar",  FLD_TYPE_ALPHA,  0,  StoreInfo,  strlen(StoreInfo),
Line4-          . . . . . .
Line5-          NULL);
```

POS Hook Functions

Below is a listing of the POS Hook functions:

hook_init

Description: Initialization of the hook application.

Input(s): None.

Output: integer OK.

hook_idle

Description: This hook is called once every second by the post application at all times (except when very busy.

Input(s): None.

Output: None.

hook_before_no_sale_operation

Description: This hook is executed just before the start of the next transaction. To be more precise, the hook is executed after the drawer is closed following the previous operation, for example: a sale transaction, pickup, sign-on, etc.

Input(s): None.

Output: integer OK.

hook_at_end_of_transaction

Description: This hook is executed at the end of each sales transaction, but before the printing of the trailer message, and before sending transactions to the host.

Input(s): char execute.

Globals: SALE_TTL structure is defined in 'POSTYPE.H'.

Output: integer OK.

hook_sign_secure_wait

Description: This hook is executed before sign-on, sign-off, start/stop secure mode and start/stop wait mode.

Value	Meaning
KEY_SIGN_ON	Before sign-on
KEY_SIGN_OFF	Before sign-off
KEY_BREAK	Before or after start of secure mode, depending on the value of the field pp.break_mode.
KEY_WAIT	Before or after start of wait mode, depending on the value of the field pp.wait_mode.

Input(s): short integer func.

Output: integer OK, or BAD to terminate function execution.

hook_after_total_key

Description: This hook is executed immediately after the Subtotal key is pressed, but before any operation is executed.

Input(s): struct sale_ttl_ * SALE_TTL.

Output: integer OK.

hook_init_sale_entry

Description: This hook is executed at the beginning of item entry is completed. Hook can affect application returning OK or BAD. Hook has no affect by return OK_CONTINUE.

Input(s): struct sale_entry_ *SE.

(Defined in 'POSTYPE.H')

Output: short OK, BAD or OK_CONTINUE.

hook_pre_sale_entry

Description: This hook is executed just before item entry begins. Hook can affect application returning OK or BAD. Hook has no affect by return OK_CONTINUE.

Input(s): struct sale_entry_ *SE.

(Defined in 'POSTYPE.H')

Output: short OK, BAD or OK_CONTINUE.

hook_ post_sale_entry

Description: This hook is executed just after item entry is completed. Hook can affect application returning OK or BAD. Hook has no affect by return OK_CONTINUE.

Input(s): struct sale_entry_ *SE.

(Defined in 'POSTYPE.H')

Output: short OK, BAD or OK_CONTINUE.

hook_plu_before_read

Description: This hook is executed after entering or scanning the item but before the item information is read from the item file.

The return from the hook will determine if PLU sale execution is continued or terminated.

Input(s): struct sale_entry_ *SE - Defined in POSTYPE.H'.

struct PLU_ *plu - Defined in POSSALE.H'.

Output: integer OK, or BAD to terminate function execution.

Remarks: The field code in the PLU structure is packed in BCD format. To unpack, use the following procedure:

```
char plu_code[14];
unpak ( plu_code, plu->code, 14 );
```

hook_plu_after_read

Description: This hook is executed after the item information is read, but before the information is processed in any way.

The return from the hook will determine if the PLU sale execution is continued or terminated.

Input(s): struct sale_entry_ *SE - Defined in POSTYPE.H'.

struct PLU_ *plu - Defined in 'POSSALE.H'.

short integer *rc - A pointer to the I-O return code, can be OK or !OK.

Output: integer OK, or BAD to terminate function execution.

hook_plu_after_completion

Description: This hook is executed after the item transaction is competed.

Input(s): struct sale_entry_ *SE - Defined in POSTYPE.H'.

struct PLU_ *plu - Defined in 'POSSALE.H'.

struct BROM_ *brom - Defined in 'POSSALE.H'

(Active promotion structure).

Output: integer OK.

hook_ brom_before_read

Description: This hook is executed just before promotion information is read from the BROM file.

Input(s): struct sale_entry_ *SE - defined in 'POSTYPE.H'

struct PLU_ *PLU – defined in 'POSSALE.H'

struct BROM_ *brom - defined in 'POSSALE.H'

Output: integer OK or BAD to terminate function execution.

Remarks: The return from the hook will determine if the promotion on the item will take place or not. (OK/BAD)

hook_ brom_after_read

Description: This hook is executed just after promotion information is read from the brom file.

Input(s): struct sale_entry_ *SE - defined in 'POSTYPE.H'

struct PLU_ *plu - defined in 'POSSALE.H'

struct BROM *brom - defined in 'POSSALE.H'

short integer *rc - a pointer to a return code integer.

Output: OK or BAD to terminate function execution.

Remarks: The return from the hook will determine whether the promotion on the item will take place or not. (OK/BAD).

If *rc !=OK, this means that the item was not found.

hook_brom_redc_after_completion

Description: This hook is executed after reduction is completed. Only reductions have an after completion hook, because promotions and offers are executed as a special PLU sale. Therefore the 'PLU after completion' hook covers this purpose.

Input(s): struct sale_entry_ *SE

struct PLU_ *plu

struct BROM_ *brom

Output: integer OK.

hook_dep_before_read

Description: This hook is executed after entering a department, but before the department information is read from the item file.
The return from the hook will determine if the department sale execution is continued or terminated.

Input(s): struct sale_entry_ *SE - Defined in 'POSTYPE.H'.

short integer *dep_no - Department number.

Output: integer OK, or BAD to terminate function execution.

Remarks: If the department value *dep_no is change, the sale will take affect at the new department.

hook_dep_after_read

Description: This hook is executed after the item information is read, but before the information is processed in any way.

The return from the hook will determine if the department sale execution is continued or terminated.

Input(s): struct sale_entry_ *SE - Defined in 'POSTYPE.H'.

struct DEP_ *dep - Defined in 'POSSALE.H'.

short integer *rc - A pointer to the I-O return code, can be OK or !OK.

Output: integer OK, or BAD to terminate function execution.

Remarks: The department code in the department structure is packed in BCD format. To unpack, use the following procedure:

```
int dep_no;
dep_no =  bcd_2_int (dep->no );
```

hook_dep_after_completion

Description: This hook is executed after the department sale is competed.

Input(s): struct sale_entry_ *SE Defined in 'POSTYPE.H'.

struct DEP_ *dep Defined in 'POSSALE.H'.

Output: integer OK.

hook_disc_before_read

Description: This hook is executed after selecting the discount, but before the discount information is read from the discount file.

The return from the hook will determine if the discount execution is continued or terminated.

Input(s): short integer *disc_no - A pointer to the discount number.

Output: integer OK, or BAD to terminate function execution.

Remarks: If the discount value *disc_no is change, the sale will take affect at the new discount.

hook_disc_after_read

Description: This hook is executed after the discount information is read, but before the information is processed in any way. The return from

the hook will determine if discount execution is continued or terminated.

Input(s): struct DISCOUNT_ *discount - Defined in 'POSSALE.H'

Short integer *rc - A pointer to the I-O return code, can be OK or !OK.

Output: integer OK.

hook_disc_after_completion

Description: This hook is executed after the discount is completed.

Input(s): struct discount_entry_ *DE - Defined in 'POSTYPE.H'.

struct DISCOUNT_* discount - Defined in 'POSSALE.H'.

Globals: None.

Output: integer OK.

hook_retn_before_read

Description: This hook is executed after selecting the return, but before the return information is read from the returned file.

The return from the hook will determine if the return execution is continued or terminated.

Input(s): short integer *retn_no - A pointer to a return code integer.

Output: integer OK.

Remarks: If the return value *retn_no is change, the sale will take affect at the new return number.

hook_retn_after_read

Description: This hook is executed after the return information is read, but before the information is processed in any way.

The return from the hook will determine if the return execution is continued or terminated.

Input(s): struct RETURN_ *retn - Defined in'POSSALE.H'.

short integer *rc - A pointer to the I-O return code, can be OK or !OK.

Output: integer OK, or BAD to terminate function execution.

hook_tndr_before_read

Description: This hook is executed after selecting the tender, but before the tender information is read from the tender file.

The return from the hook will determine if the tender execution is continued or terminated.

Input(s): short integer *tender_no - A pointer to the tender number.

Globals: None.

Output: integer OK or BAD to terminate function execution.

Remarks: If the tender value *tender_no is change, the sale will take affect at the new tender number.

hook_tndr_after_read

Description: This hook is executed after the tender information is read, but before the information is processed in any way.

The return from the hook will determine whether the tender execution is continued or terminated.

Input(s): struct tender_entry_ *TE

struct TENDER_*tender - Defined in 'POSMEDIA.H'.

short integer *rc - A pointer to the I-O return code, can be OK or !OK.

struct sale_ttl_ * SALE_TTL.

Output: integer OK.

Remarks: The tender code in the tender structure is packed in BCD format. To unpack, use the following procedure:

```
int tender_no;
tender_no  = bcd_2_int (tender  -
>no );
```

hook_tndr_after_completion

Description: This hook is executed after tendering is competed.

Input(s): struct tender_entry_*TE_- Defined in 'POSTYPE.H'.

struct TENDER_*tender – Defined in' POSMEDIA.H'.

Output: integer OK.

hook_freq_shopper

Description: This hook is executed after the frequent shopper entry, but before it is validated.

The return from the hook will determine whether to continue, or re-enter the frequent shopper entry.

Input(s): struct ppm_entry_*hook_ppm_entry - Defined in 'POSTYPE.H'.

Globals: None.

Output: integer OK, or BAD to terminate function execution.

hook_bypass_check_signature

Description: This hook is executed in both regular and refund transactions just before signature verification is bypassed.

The return from the hook will determine whether to continue or terminate the tender execution.

Input(s): None.

Output: integer OK, or BAD to terminate function execution.

Note:

ISS45 offers two signature *verification* methods: (1) Non-Captured Signature Verification - The POS displays a “Verify Signature” message to the cashier to remind the cashier to check the customer’s card signature with the signature on the multi-part receipt. (2) Signature Capture Verification - The POS displays the captured signature from the EFT terminal on the Cashier Display with the option to accept or reject the signature. The hook_bypass_check_signature hook is used with the Tender Maintenance Screen’s “Max. Bypass Amt for Signature check:” field. This field defines the maximum credit card tender amount for bypassing signature verification. The field default (0.00) indicates that the signature check is never bypassed (assuming the transaction meets all other requirements for signature). If the hook_check_signature hook (below) is enabled, it will not be executed if the “Max. Bypass Amt for Signature check:” amount has not been exceeded.

hook_check_signature

Description: This hook is executed just before the 'check signature' prompt.

The return from the hook will determine whether to continue or terminate the tender execution.

Input(s): None.

Output: integer OK, or BAD to terminate function execution.

hook_function

Description: This hook is used for user hook functions.

The user should put a hook function into the keyboard layout or into the POS menu. Using a hook function will result in execution of this hook with the hook function number.

The return from the hook will affect the operation of the register.

If the hook terminates with the OK return code, then the POS will try to execute the function value that exists in the argument 'ppm_entry->code`. It is allowed to put values of POS FUNCTIONS like X_READ or NO_SALE.

If the hook terminates with the BAD return code, then POS continues normally.

Input(s): struct ppm_entry_*hook_ppm_entry - Defined in 'POSTYPE.H'.

Output: integer OK, or BAD (see function description).

Remarks: The hook function number is hook_ppm_entry->code -KEY_HOOK_ST + 1;

It is possible to transfer keyed-in numbers into the hook function in the 'long' argument 'ppm_entry->result`.

KEY_HOOK_ST (hook function start) = 701

KEY_HOOK_END (Last hook function key) = 750

hook_before_maintenance

Description: This hook is executed just before every maintenance execution. The return from the hook will determine whether to continue execution of this maintenance (OK/BAD).

Input(s): union maint_ * maint

Output: integer OK.

Remarks: All maintenance types are defined in 'POSMAINT.H'.

The general maintenance is defined as follows:

Byte opcode: maint->mnt_rec.opcode

Datat 96 bytes: maint->mnt_rec.data

hook_after_maintenance

Description: This hook is executed just after every maintenance execution.

Input(s): union maint_ *maint

Output: integer OK.

Remarks: All maintenance types are defined in 'POSMAINT.H'.

The general maintenance is defined as follows:

Byte opcode: maint->mnt_rec.opcode

Datat 96 bytes: maint->mnt_rec.data

hook_after_total_key_and_prom

Description: This hook is executed just after the Subtotal key is pressed and after any operation is executed.

Input: struct sale_ttl_ *SALE_TTL.

Output: None.

hook_after_read_scanner

Description: This hook is executed after data arrived from the scanner, and it’s in the global scanner buffer of the WinPos. The user

has the ability to change this data, if so the new data will be updated after the function has ended.

Input: char *scanner_buf

Output: None.

hook_after_read_msr

Description: This hook is executed after data arrived from the MCR (Magnetic Card Reader). This function gives the user the ability to view/change the 3 tracks data. The data will be updated after the function has ended.

Input: struct track_ *track1,

struct track_ *track2,

struct track_ *track3

The struct track_ is defined in: posmedia.h

Output: None.

hook_before_save_transaction

Description: This hook is executed before the actual save ticket process begins. The user can view/change the SALE_TTL struct which holds the current ticket balancing and data.

Input: struct sale_ttl_ *SALE_TTL

The struct sale_ttl_ is defined in: postype.h

Output: If the function returns something other than OK (0), then the save process will be terminated. Default return to this function is OK (0).

hook_after_save_transaction

Description: This hook is executed after the Save Ticket process has ended. The user can view/change the SALE_TTL struct which holds the current ticket balancing and data.

Input: struct sale_ttl_ *SALE_TTL

The struct sale_ttl_ is defined in: postype.h

Output: None.

hook_before_recall_transaction

Description: This hook is executed before the actual Read process begins. The user receives customer id string, length of the customer id string, pointers to the entered ticket number and the entered pos number; the user has the ability to change the pos number in order to recall any other ticket that is available.

Input: char *cust_id,

short length,

long *ticket_no,

long *pos_no

Output: If the function returns something other than OK (0), then the recall process will be terminated. Default return to this function is OK (0).

hook_after_recall_transaction

Description: This hook is executed after the recall ticket process has ended. The user can view/change the SALE_TTL struct which holds the current ticket balancing and data.

Input: struct sale_ttl_ *SALE_TTL

The struct sale_ttl_ is defined in: postype.h

Output: None.

hook_before_eod

Description: This hook is executed after EOD (before WinPOS rerun and after all communication ports have been closed) in order to allow sending data to peripherals such as printer, etc.

Input: None.

Output: None.

hook_after_tax_exempt_scan

Description: This hook is executed after the prompt for the Tax Exempt No. is presented on the POS terminal to allow validation of the customer maintained Tax IDs.

Input: struct ppm_entry_ *ppm_entry.

Output: integer OK, or BAD to terminate function execution.

hook_before_age_verify

Description: This hook is executed before a customer birth date is entered.

Input: char bday_prompt.

char bypass_cc

short interger *month

short *day

short *year

Output: integer OK, or BAD to terminate function execution.

Functions, Structures & Definitions

Available Function Calls

The following functions are just a few examples of those available. For a full list of functions refer to the 'POSDEC.H' file.

unpak

Description:	This function unpacks a number from BCD format to ASCII format.
Input(s):	unsigned char *u_str - Unpacked field.
	unsigned char *p_str - Packed field.
	integer n - The number of digits in the unpacked field.
Output:	None.

pak

Description:	This function packs a number from ASCII format to BCD format.
Input(s):	unsigned char *u_str - Unpacked field.
	unsigned char *p_str - Packed field.
	integer n - The number of digits in the unpacked field.
Output:	None.

bcd_2_int

Description:	This function unpacks a number from BCD format to INT format.
Input(s):	char *bcd - Packed field.
Output:	(integer)Value of the BCD field.

Hook_transaction

Description:	This function writes a user hook transaction.
Input(s):	unsigned char sub_opcode.

	Use only **0x01** that defines the transaction as a user hook.
	char *data-A 42 character long string that holds the transaction data.
Output:	None.

system_key

Description:	Wait for a keyboard key and background processing continues.
Input(s):	None.
Output:	(unsigned integer) value of the pressed key.

WPFUNC_get_global_value

Description:	Obtain variables stored in the POSW32 application.
Input(s):	char *desc. Name of variable to retrieve.
Output:	None.

WPFUNC_set_global

Description:	Allows setting of a POSW32 variable.
Input(s):	char *desc. Name of variable to set.
	char value. Value to set variable
Output:	None.

WPFUNC_set_global_value

Description:	Allows setting of a POSW32 variable.
Input(s):	char *desc. Name of variable to set.
	long value. Value to set variable
Output:	None.

WPFUNC_ calc_last_trans_ptr

Description:	This function will calculate the last txn in a current ticket. Use this function when

	you need to parse the current txns in a ticket.
Input(s):	None.
Output:	None.

WPFUNC_ read_prev_trans

Description:	This function is used to read the previous txn in the current ticket. Use this function when parsing the current txns in a ticket.
Input(s):	struct file_parm_ *f_parm, union trans_ *trs.
Output:	None.

WPFUNC_ MENU_state

Description:	This function is used to set the current menu state.
Input(s):	short state.
State Values:	STATE_CLEAR
	STATE_YESNO
	STATE_ALFA
	STATE_YESNO_ACCEPT_ABORT
	STATE_BREAK
	STATE_PASSWORD
	STATE_SUPERVISOR
	STATE_CLEAR_ABORT
	STATE_ENTER_ABORT
	STATE_VENDOR_COUPON
	STATE_STORE_COUPON
	STATE_BONUS_COUPON
	STATE_VENDOR_COUPON_AMT
	STATE_STORE_COUPON_AMT
	STATE_BONUS_COUPON_AMT
	STATE_TENDER_CORECTION
	STATE_CANCEL_CLEAR

STATE_TENDER_CORECTION2

STATE_CASHENTER

STATE_CANCEL

STATE_CASHBACK

STATE_UPDOWN_CLEAR

STATE_PREPAY_PURCHASE

STATE_FUEL_FOR_CHANGE

STATE_TOUCH_SCREEN_IDLE

STATE_MENU_ENTER_ABORT_DEP

STATE_MENU_MANUAL_ENTRY_CANCEL

STATE_PRESET_PUMP

STATE_PGUP_PGDOWN_CLEAR

STATE_MENU_RETURN

STATE_MENU_PRICE_INQUIRY

STATE_MENU_PRICE_OVERRIDE

STATE_MENU_NOT_ON_FILE

Output: None.

WPFUNC _MENU_resume_state

Description: This function is used to resume the POS to its previous menu state after the WPFUNC_MENU_state function is executed.

Input(s): None.

Output: None.

WPFUNC _ force2_key_buffer

Description: This function is used to force a key stroke into the key buffer.

Input(s): short key.

Output: None.

WPFUNC _ sell_plu

Description:	This function is used to call the POS sell_plu function.
Input(s):	struct sale_entry_ *SE.
Output:	None.

WPFUNC _ special_write_transaction

Description:	This function will write a txn to the transact.qdx file.
Input(s):	union trans_ *special_trs, short.
	send_option, unsigned char tail_option.
Output:	None.

WPFUNC _ cashier_totals_tender

Description:	This function will update the media totals per tender.
Input(s):	struct tender_entry_ *TE, struct TENDER_ *tender.
Output:	None.

WPFUNC _ get_tender

Description:	This function will retrieve the tender definition based on the tender number.
Input(s):	short tender_no, struct TENDER_ *tender.
Output:	None.

WPFUNC _ get_tax

Description:	This function will retrieve the tax definition based on the tax number.
Input(s):	short tax_no, struct TAX_ *tax.
Output:	None.

WPFUNC _ calc_FS_bal_due

Description:	This function will return the current food stamp balance due (Vales de Alimentacion).
Input(s):	None.
Output:	long, Current Food Stamp total due.

WPFUNC _ get_plu

Description:	This function will return the PLU information for the PLU number entered.
Input(s):	struct PLU_ *plu, char comm_read.
Output:	Short, 0 = OK.

Structure Definitions

The following structure definitions and descriptions are only some examples. For a full list of structure definitions refer to the include files

Sale_ttl

Defined in:	'POSTYPE.H'.

struct sale_entry_

Defined in:	'POSTYPE.H'.
SE->ppm_entry.code	Entry source: KEY_ENTER, KEY_SCANNER & KEY_PRESET.
SE->count	Item count.
SE->weight	Item weight (ounces/grams).
SE->decimal_count	Item decimal count (millimeters).
SE->return_item	Next item is returned type 'SE->return_type'.
SE->cancel_item	This is last item cancellation.
SE->subtract_item	Subtract the entered item.
SE->tax_reverse	Sell next item and reverse tax.
SE->FS_reverse	Sell next item and reverse food stamp.
SE->inquery	Item is inquired first, before sold.

struct PLU_

Defined in:	'POSSALE.H'.

struct BROM_

Defined in:	'POSSALE.H'.

struct DEP_

Defined in:	'POSSALE.H'.

struct DISCOUNT_

Defined in:	'POSSALE.H'.

struct discount_entry_

Defined in:	'POSTYPE.H'.
DE->discount_amount	Discount value.
DE->discount_percent	Discount percent.
DE->return_item	The discount is on return item.
DE->cancel_item	The discount is canceled.
DE->subtract_item	The discount is subtracted.
DE->plu_dep_amount	PLU or DEPT. discountable value.
DE->extra_amount	Extra discountable value, affected by any kind of PLU/DEPT. operation, like discount or promotion. The discountable amount is, therefore, calculated in the following way: Discounttable_value =DE->plu_dep_amount + DE->extra_amount;

struct RETURN_

Defined in: 'POSSALE.H'.

struct TENDER_

Defined in: 'POSMEDIA.H.'

struct tender_entry_

Defined in:	'POSTYPE.H'.
TE->tender_amount	Tender amount (in local currency).
TE->foreign_amount	Foreign currency value.
TE->foreign_rate	Foreign currency rate.
TE->account	Account number.
TE->auth_no	Authorization number.
TE->exp_date_month	Expired month.
TE->exp_date_year	Expiry year.
TE->cancel_tender	Canceled tender.
TE->subtract_tender	Subtracted tender.
TE->change	This is change.
TE->MCR_used	Swipe card was used for this tender.

struct ppm_entry_

Defined in: 'POSTYPE.H'.

Defined Values

Defined values are in the include files 'POSDEF.H' & 'POSPPM.H'.

'POSDEF.H' holds the POS application definitions such as: SALE, TENDER**,** etc.

'POSPPM.H' holds the general application definitions such as: KEY_MENU, KEY_YES, etc.

Get Global Variables

The variables below can be read using the WPFUNC_get_global_value function.

G Structure

G_OP_2x20_Active	G.OP_2x20_Active
G_EFT_IDLE_REQ	G.CUSTOMER_DISP
G_EFT_SIMULATE_RESPONSE	
G_SIGNATURE_CAPTURE_SUPPORTED	
G_SYSTEM_IS_READY	G_KEYB_START_Y
G_RECOVERING_TICKET	
	G_PROCESSING_SELF_CHECKOUT
G_SCOT_ASSIST_MODE	G_SELF_CHECKOUT_ATTACHED
G_VOID_TICKET	G_FUEL_CLOSE_PREPAY
G_HOOK_ALPHA_TOUCH_KEYBOARD	
G_EFT_MANUAL_EOD	G_EFT_MANUAL_PRE_EOD
G_EFT_MANUAL_DEBIT	G_EFT_MANUAL_CREDIT
G_FISCAL_EOD_RETURN	G_FISCAL_IS_ACTIVE
G_FISCAL_NOPRINT_ACTIVE	G_NONFISCAL_COMMENT_PRINT
G_Accumulate_Food_Stamp_Tenders	
G_save_process	G_SmartCard_ID

C Structure

C_SPECIAL_ISS45_ENTRY C_MSG_IS_WAITING
C_RCPT_EJ_ENABLE C_ABORT_BUFFER_PRINTING
C_SALE_EXECUTION C_REORGANIZE_RECEIPT
C_STS_LINE C_MAIN_SALE_ENTRY
C_MAIN_TENDER_ENTRY C_RECOVER_TRANSACTION
C_RECOVER_CURRENT_TRANS C_REMOTE_SIGN_OFF_ALLOWED
C_REMOTE_SIGN_ON_ALLOWED
C_DONT_INC_TICKET_NO C_REPRINT_LAST_TICKET
C_WIC_ENDORSE_PRINT C_MAINT_EXECUTION
C_MOVE_CCMS_MSG_TO_BKGD

SALE TTL Structure

SALE_TTL_ticket_amount SALE_TTL_media_total
SALE_TTL_tender_correction SALE_TTL_ticket_amount_before_tax
SALE_TTL_ticket_tax_value SALE_TTL_total_saving
SALE_TTL_ccms_customer SALE_TTL_offline_ccms_customer
SALE_TTL_multisaver_total SALE_TTL_total_points
SALE_TTL_min_purch_amount SALE_TTL_freq_shop
SALE_TTL_eft_last_tender_authorized
SALE_TTL_scanner_used SALE_TTL_return_ticket
SALE_TTL_ticket_freq_shop SALE_TTL_tax_exempt
SALE_TTL_print_bal_due SALE_TTL_trans_disc_executed
SALE_TTL_ticket_discount_percent SALE_TTL_eft_sa_print_reject
SALE_TTL_ticket_items SALE_TTL_tender_purchase
SALE_TTL_recover_ticket SALE_TTL_recall_ticket
SALE_TTL_recalled_ticket_no SALE_TTL_customer_bday_day
SALE_TTL_recalled_pos_no SALE_TTL_bonus_buy_ttl
SALE_TTL_customerinfo_name SALE_TTL_customerinfo_fiscalnum
SALE_TTL_customerinfo_phone SALE_TTL_customerinfo_zone

SALE_TTL_Pangui_Promotion_Total
SALE_TTL_reprocess_screen_on

CMOS Structure

PP_pos_no	PP_cashier_no
PP_pos_status	PP_training_mode
PP_ticket_no	PP_recall_recover
PP_signon_order	PP_last_control_server
PP_trans_pointer	PP_trans_count
PP_future_trans_count	PP_trans_fifo_size
PP_trans_pointer	PP_trans_count
PP_future_trans_count	PP_trans_fifo_size
PP_reprocess_in_progress	PP_late_swipe_recover
PP_reprocess_ccms_prom_only	PP_fiscal_ticket_number
PP_fiscal_serial_number	PP_RUC_ticket_count

POS_PARAM Structure

POS_PARAM_supervisor_privilege
POS_PARAM_use_FS_num_for_check_tnd
POS_PARAM_RR_buffer_printing POS_PARAM_RR_main_sort
POS_PARAM_RR_second_sort POS_PARAM_RR_sort_receipt
POS_PARAM_RR_combine_sales
POS_PARAM_RR_combine_cancel_subt
POS_PARAM_RR_group_all_coupons
POS_PARAM_RR_tax_report_before_bal_due
POS_PARAM_RR_net_price_on_system
POS_PARAM_RR_group_complex_promotions
POS_PARAM_RR_special_format_for_negatives
POS_PARAM_RR_special_format_for_savings
POS_PARAM_RR_special_format_for_net_price
POS_PARAM_RR_special_format_for_coupons
POS_PARAM_RR_special_format_for_additional_savings
POS_PARAM_RR_combine_price_compare

POS_PARAM_RR_combine_member_promo

POS_PARAM_print_net_and_gross_price_on_same_line

POS_PARAM_opt_savings_present_as_negative

POS_PARAM_opt_savings_detail_list

POS_PARAM_opt_savings_marketing_msg

POS_PARAM_total_sales_text POS_PARAM_net_price_sale_text

POS_PARAM_net_price_free_textPOS_PARAM_total_coupon_text

POS_PARAM_additional_savings_text

POS_PARAM_additional_savings_total_text

POS_PARAM_max_no_wic_items_print_on_check

POS_PARAM_offline_credit_tender
POS_PARAM_offline_ebt_tender

POS_PARAM_get_ccms_msg2_in_bkgd

POS_PARAM_OTP_Timeout POS_PARAM_cashier_file_on_pos

POS_PARAM_allow_courtesy_card_for_sec_id_wo_member_card

POS_PARAM_return_items_trigger_promotions

POS_PARAM_CC_for_overrideable_decline

POS_PARAM_Storeline_EPS_provider

POS_PARAM_Storeline_EPS_file_directory

POS_PARAM_opt_capture_GEN_NUM_for_manual

POS_PARAM_opt_allow_offline_gift_card_activation

POS_PARAM_gift_card_activation_tender_no

POS_PARAM_private_card_tender_no

POS_PARAM_plug_flag2_opt_iss45_eps

POS_PARAM_plug_flag2_signature_capture

POS_PARAM_private_credit_tender_number

POS_PARAM_allow_courtesy_card_for_sec_id_wo_member_card

POS_PARAM_RR_enhanced_net_price_printing

POS_PARAM_RR_print_promotion_savings_line

POS_PARAM_RR_print_regular_price_line

POS_PARAM_RR_bonus_buy_savings_text

POS_PARAM_use_catalina21_message_format

POS_PARAM_get_catalina_2_1_port
POS_PARAM.catalina_2_1_ip

POS_PARAM_display_member_name_eft

POS_PARAM_apply_idle_locking_mode_for_pin_pad

POS_PARAM_customized_EFT_messaged_on_cashier_display

POS_PARAM_immediate_processing_of_voice_authorization

POS_PARAM_auto_alpha_keyboard_prompts_for_alphanumeric_EFT_fields

POS_PARAM_CC_dual_entry_of_manual_check_account

POS_PARAM_CC_dual_entry_of_manual_check_number

POS_PARAM_CC_dual_entry_of_manual_secondary_ID

POS_PARAM_CC_for_viewing_check_velocity_data

POS_PARAM_currency_sign POS_PARAM_negative_sign

POS_PARAM_CC_dont_allow_gift_card_reuse

POS_PARAM_main_eft_dept

POS_PARAM_bad_account_file_on_pos

SYS_PARAM Structure

SYS_PARAM_weight_type	SYS_PARAM_store_no
SYS_PARAM_divis_no	SYS_PARAM_weight_factor
SYS_PARAM_ccms_pc	SYS_PARAM_dec_point

PD Structure

PD_EFT_PORT	PD_EFT_BAUD
PD_CUST_WIDTH	PD_QDX_CCMS_FILE
PD_QDX_TRANS_FILE	

LOCN Structure (POS Configuration)

LOCN_auth_port	LOCN_auth_baud
LOCN_host_ip_address	LOCN_host_port_number
LOCN_eft_app_param_version	LOCN_eft_app_prog_version
LOCN_pinpad_unit_type	LOCN_apt_port
LOCN_apt_baud	LOCN_interfaces
LOCN_pos_accountability_virtual_pos	
LOCN_storeline_eps_pinpad_type	LOCN_authorization

Set Global Variables

The variables below can be set using the WPFUNC_set_global and WPFUNC_set_global_value functions.

G Structure

G_OP_2x20_Active	G_EFT_IDLE_REQ
G_SIGNATURE_CAPTURE_SUPPORTED	
G_HOOK_ALPHA_TOUCH_KEYBOARD	
G_SmartCard_St	G_SmartCard_Init_Pts
G_SmartCard_ID	G_SmartCard_Init_Pts
G_SmartCard_Gain_Pts	G_SmartCard_Acc_Pts
G_SmartCard_Phone	G_SmartCard_Number
G_SmartCard_Used_Pts	G_SmartCard_Virt_Pts
G_SmartCard_Path	G_NONFISCAL_COMMENT_PRINT
G_SCOT_ASSIST_MODE	

C Structure

C_SPECIAL_ISS45_ENTRY	C_MSG_IS_WAITING
C_ABORT_BUFFER_PRINTING	C_DONT_INC_TICKET_NO
C_OPT_CAT_PLUG	C_OPT_APT_PLUG
C_MOVE_CCMS_MSG_TO_BKGD	
C_OPTR_IS_DEACTIVATED	C_SALE_EXECUTION

SALE TTL Structure

SALE_TTL_ticket_amount	SALE_TTL_media_total
SALE_TTL_tender_correction	SALE_TTL_ticket_amount_before_tax
SALE_TTL_ticket_tax_value	SALE_TTL_total_saving
SALE_TTL_ccms_customer	SALE_TTL_offline_ccms_customer
SALE_TTL_multisaver_total	SALE_TTL_total_points
SALE_TTL_eft_last_tender_authorized	
SALE_TTL_return_ticket	SALE_TTL_print_bal_due
SALE_TTL_eft_sa_print_reject	SALE_TTL_update_row_value

SALE_TTL_customerinfo_name	SALE_TTL_customerinfo_fiscalnum
SALE_TTL_customerinfo_phone	SALE_TTL_customerinfo_zone
SALE_TTL_Travel_Voucher_Ref_Num	
SALE_TTL_Credit_Number	SALE_TTL_Credit_Amount
SALE_TTL_RUC_Amount	SALE_TTL_Warranty_Cust_ID
SALE_TTL_Warranty_Svc_Plan	SALE_TTL_Warranty_Duration
SALE_TTL_Warranty_Item_UPC	SALE_TTL_Warranty_Item_Desc
SALE_TTL_Warranty_UPC	

CMOS Structure

PP_pos_no	PP_cashier_no
PP_pos_status	PP_training_mode
PP_ticket_no	PP_last_control_server
PP_trans_pointer	PP_trans_count
PP_future_trans_count	PP_trans_fifo_size
PP_reprocess_ccms_prom_only	PP_RUC_ticket_count

pd Structure

PD_EFT_PORT	PD_EFT_BAUD
PD_QDX_CCMS_FILE	PD_QDX_TRANS_FILE

POS_PARAM Structure

POS_PARAM_supervisor_privilege

POS_PARAM_use_FS_num_for_check_tnd

POS_PARAM_OTP_Timeout

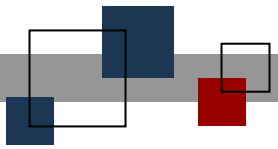
SYS_PARAM Structure

SYS_PARAM_store_no

LOCN Structure

LOCN_auth_port	LOCN_auth_baud
LOCN_host_ip_address	LOCN_host_port_number
LOCN_eft_app_param_version	LOCN_eft_app_prog_version

LOCN_pinpad_unit_type



StoreNext Retail Technologies LLC
6100 Tennyson Parkway, Suite 130
Plano, Texas 75024

P/N 89000561
PIN 45001/116