Infor Integration 6.2

Developer's Guide for Application Function Server

Publication Information

Release: Infor10

Publication date: October 4, 2011

Document code: U8627B US

Contents

About this guide

This document is a Developer’s Guide designed to serve as a Reference Guide for the Application Function Server (AFS). This document covers both the architecture of, and programming with, the AFS. Upon completion of this document, the reader will have the necessary knowledge to modify existing AFS code and to create new AFS code.

The generic term Infor ERP refers in this guide to the following ERP systems: ERP Baan IV, ERP Baan5.x, and ERP Enterprise (LN). If a particular release is meant, the actual product name and version number are specified.

Intended audience

This guide is intended for developers and system administrators.

Related documents

You can find the documents in the product documentation section of Infor365, as described in "Contacting Infor".

- Infor Open Architecture Adapter Suite 2.7– Developer's Guide for ERP Baan IV, ERP Baan and ERP LN Servers (U8638 US)
- ERP Baan IVc - BOI Builder Developer's Guide (U7194 US).

Contacting Infor

If you have questions about Infor products, go to Infor365 Online Support at http://www.infor365.com.

If we update this document after the product release, we will post the new version on Infor365. We recommend that you check this Web site periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Chapter 1 Introduction

1

This document is designed to serve as a Reference Guide for the Application Function Server (AFS). This document covers both the architecture of, and programming with, the AFS. After reading this document, the reader will have the necessary knowledge to modify existing AFS code and to create new AFS code.

This document also describes the restrictions for Baan 4GL application sessions, which the user must bear in mind during the development and maintenance of these Baan 4GL application sessions. For this reason, you must also use this document when you build new (standard) applications, even if it is not clear yet that the AFS is going to be used together with those new applications.

Scope

This document is intended for experienced Infor ERP programmers only. No attempt is made to explain Baan 4GL programming or Infor ERP architecture outside of what is required for the AFS.

The document applies to ERP Baan IV, ERP Baan5.x, and ERP Enterprise (LN)[1] in general. However, the implementation and the use of the AFS can differ. When applicable, these differences are noted in the text.

Definitions, acronyms, and abbreviations

Term	Description
AFS	Application Function Server: Not to be confused with the DDC, or Distributed Data Collection, Function Servers, this is another type of technology.

[1] Because the AFS is related to the Tools version, this document applies to the Tools versions B40c (for Baan IV), B50b, 7.1a and 7.3a (for ERP Baan), and Infor Enterprise Server (LN). For future versions and releases this document may need modifications.

Term	Description
AFS-DLL	An Infor ERP DLL that contains the calls to the API-handler for a specific Baan 4GL application session.
API	Application Programming Interface.
API-handler	The program that serves outside the application to call the functionality of the Baan 4GL application sessions.
BCBE	Baan Connection Basic Edition: This technology is replaced by OpenWorldX, which is now called the Open Architecture Adapter Suite.
BOI	Business Object Interface.
DAL	Data Access Layer (DAL): The software layer in Infor ERP systems that contains all data manipulation rules, such as the authorization for modifying, removing, or adding data, and all related constraints. This software layer also includes all rules required to maintain data consistency.
ERP	Enterprise Resource Planning (ERP) is a business management system that integrates all aspects of the business, including planning, manufacturing, sales, and marketing.
FS	Function Server: This term is sometimes used to refer to AFS.
Infor ERP	Infor ERP is a generic term for the following ERP systems of Inforl: Infor ERP Baan IV, Infor ERP Baan5.x, and Infor ERP Enterprise (LN).

Chapter 2 Application Function Server

2

One of the preferred methods of integration between Infor ERP systems (Baan IVc, Baan5.x, and ERP Enterprise (LN) and third-party products is by means of Business Object Interfaces or BOIs. BOIs provide an Application Programming Interface (API) for the ERP Baan business logic and run in the context of BCBE, OpenWorldX, or Open Architecture Adapter Suite.

The high-level interfaces that the BOIs provide cannot be used directly against the ERP Baan business logic due to the ERP Baan architecture. The business logic is implemented by means of sessions, which are primarily user-interface based. Programmatic access to sessions is provided by means of a low-level message protocol. This message protocol is encapsulated into a set of function primitives called the Application Function Server (AFS); these functions are located in the API-handler. The BOI code accesses the session business logic by means of the AFS. The public, high-level interfaces exposed by the BOIs are internally translated to the low level primitives understood by the AFS, which then communicates with the session logic by means of the message protocol.

Usually, you are not required to create BOIs and AFS-DLLs from scratch. Instead, you can generate BOIs with the BOI generator, for BCBE, or BOI Builder, for Open Architecture Studio. AFS-DLLs can be created with the AFS-DLL generator (ttstpcreatdll).

The use of both of these tools is described in *BOI Builder Developer's Guide* and the *Developer's Guide for ERP Baan IV, ERP Baan5.x and ERP Enterprise Servers (LN)*. You can use the generated BOIs and AFS-DLLs directly to solve straightforward integration problems where only one BOI and one AFS-DLL are involved to implement a particular interface function set. Complex integration scenarios in which multiple BOIs and AFS-DLLs are involved, for example, sales orders, require extra development in the generated BOI and AFS-DLL code. This extra development cannot be accomplished without detailed knowledge of the architecture and programming constructs involved in both the BOI and AFS-DLL code.

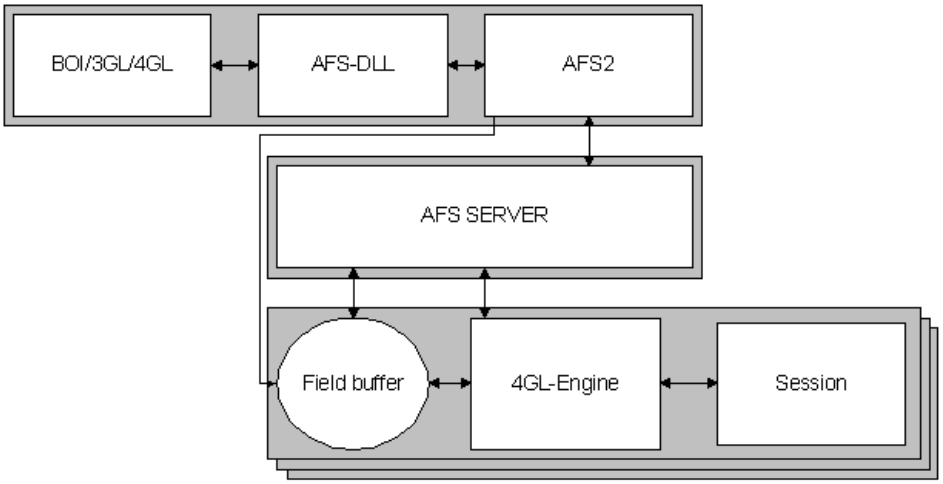
The use of an AFS-DLL is optional; the BOIs can also use the function primitives of the AFS directly. However, the function names generated in the AFS-DLL have logical names (for example, field names instead of field codes), so typing errors will be caught by the compiler. Otherwise, typing errors will be found at runtime. The AFS-DLL creates as extra layer on top of the Baan Standard Program, or 'stpapi', which decreases the performance. With the AFS-DLL, however, typing errors can be caught by the compiler, because the AFS uses logical names, for example, field names instead of field codes. Without the AFS-DLL, typing errors will not be found until runtime.

The use of AFS is not restricted to BOIs only. In addition, other Baan 3GL or 4GL programs can use the AFS to call other sessions with their business logic to perform particular tasks.

Because the BOI programming constructs are described in detail in the *Developer's Guide for ERP Baan IV, ERP Baan and ERP Enterprise Servers (LN)*, this document describes only the details of the AFS here.

Architecture

The role of the AFS in the BOI architecture is shown in this diagram:



The diagram clearly illustrates that the BOI communicates with the underlying Infor ERP session by means of the AFS.

Explanation diagram:

- BOI/3GL/4GL:
  The program that wants to access the Infor ERP session business logic.
- AFS-DLL:
  The DLL that contains wrapper functions to the AFS function primitives. This DLL is optional.
- AFS2:
  The DLL that contains the implementation of the AFS function primitives. This DLL communicates with the 4GL-Engine through a dedicated BMS protocol to enable the session to run the requested functionality.
- AFS Server:
  Server process that communicates with the 4GL-Engine through a dedicated BMS protocol to permit the session to run the requested functionality.
- Field buffer:
  The field buffer contains the fields (names and values) of the form, which are input for the

session. If the first API-call to a specific session is carried out, these fields are retrieved from the session. These fields can be table fields, but also other form fields, which are used for input.

The values from the BOI are buffered in the field buffer. If the session needs these values in the field loop of the session, the 4GL-Engine takes the values from the field buffer and makes the values available in the session.

After the session changed the field values, such as during an update or browse set, the field buffer is updated and the BOI can retrieve the new values. For more information, refer to Chapter 4, “Special issues.”

- 4GL-Engine:
  The engine for each Baan 4GL application session, which performs all form, field, event, and main table handling, etc. This engine is the same 4GL-Engine that is used to run the session in the user interface, for example, Baan Windows. While the session usually waits for user interactions, for example, filling fields, pressing buttons, etc., the session now waits for AFS commands. The variable api.mode is set to True, which causes the 4GL-Engine to handle differently. In addition, the session logic can use this predefined variable to have different behavior when called from the AFS.
- Session logic:
  The Baan 4GL application sessions that is run with the AFS. More sessions can exist if a session opens more subsessions or in the case of multi-occurrence/single-occurrence synchronization, ERP Baan 5.x and ERP Enterprise (LN) only.

  Only one instance can be active for a session. The AFS keeps track of the session instances and sends the BMS messages to the process number, which is attached to that session instance internally.

  Bear in mind that the session only works for one record at a time. For multi-occurrence forms, type 2 or 3, only one occurrence is used.

Example of using AFS

```
function long add.record.to.table(
            domain dtseno i.seno,
            domain dtkey1 i.key1,
            domain dtname i.name,
            ref string o.mess())
{
            long                                    retval
            string          dummy.msg(1)

            stpapi.put.field("dtfsa1101s000", "dtfsa100.seno", str$(i.seno))    |* 1
            stpapi.put.field("dtfsa1101s000", "dtfsa100.key1", i.key1)          |* 2
            stpapi.put.field("dtfsa1101s000", "dtfsa100.name", i.name)
            retval = stpapi.insert("dtfsa1101s000", true, o.mess)               |* 3
            if not retval then
            retval = stpapi.recover("dtfsa1101s000", dummy.msg)                 |* 4
            endif
            stpapi.end.session("dtfsa1101s000")                                 |* 5
            return(isspace(o.mess))
}
```

Explanation of the example

The numbers in the following list refer to the comments in the source code of the example:

1. The first API-call for the session, or the first argument, results in the activation of the session and the creation of the field buffer. The sections before.program, main.table.io/before.read and form1/before.form of the session are executed if present. Because it is a put.field function, the value of i.seno is put in the field buffer.
2. Other put functions on a session already activated only result in an update of the field buffer.
3. During the insert in the session, the relevant sections are run, such as choice.add.set/before.choice etc., and the field loop starts, which includes all fields of all forms. For these fields, the field sections are run, such as before.field, before.checks and check.input, or the check.property hook in the DAL for ERP Baan 5.x and ERP Enterprise (LN) only. If an error occurs, this information is returned in retval, but the message is also returned in o.mess. Because the do.save flag is set to True, the sections choice.update.db/before.choice etc. also run.
4. Recover is necessary to stop the update mode of the session if an error occurred.
5. The session must be ended; choice.end.program runs.

   This example is provided here merely offer an idea of programming with the AFS. Chapter 3, “Baan 4GL engine primitives,” which describes the AFS function primitives, offers a number of additional examples.

Structure of the AFS-DLL

The structure of the AFS-DLL can best be described by looking at an example of one. Appendix A shows the AFS-DLL for the Maintain Areas (tcmcs0145m000) session. The AFS-DLL is realized as an Infor ERP library named tcmcsf0145m000. Note the “f” in the AFS library name. The standard naming convention for an AFS-DLL is to use the name of the parent session and insert an “f” after the module code.

The AFS-DLL contains functions for setting and retrieving values for fields in the underlying session. This AFS-DLL also contains functions to add, modify, and delete records from the session. Functions for browsing are also provided. In addition, miscellaneous functions for setting answers to questions, handling subsessions, and retrieving error messages are provided.

A number of important features must be noted. First, all of the functions follow a naming convention that is obvious when the example code is perused. The naming convention must be followed when making changes to the AFS-DLL, as the BOI relies on this (if it is generated). Second, all of the AFS-DLL functions are merely a thin shell on the low-level primitives exposed by the 4GL Engine. These primitives in turn encapsulate the message protocol described previously. For more details, refer to the following chapter, which provides an in-depth description of the function primitives. An explanation of the function primitives must suffice to also describe the AFS-DLL functions.

Compilation

To compile programs that use the AFS functions, the library ottstpapihand must be linked to the program or code #pragma used dll ottstpapihand in the script.

Chapter 3 Baan 4GL engine primitives

3

In most AFS functions in which a session is mentioned as a parameter, the session is starts automatically if the session is not already running. Functions that require that the session already be started return an error if no function is called before the started session. For more information, refer to Chapter 4, “Special issues.” This implies that all subsequent function calls for the same session to be sent to the same session instance.

The sessions that the AFS starts must be 4GL sessions. You cannot run 3GL sessions without a form and a script of type 3GL (Without Std.Prog) with the AFS.

Get Field Value from session

Syntax

```
void stpapi.get.field (string session, string field, ref string value, [long element])
```

Arguments

- *session:* Name of the session on which this command is executed.
- *field*: Name of the field whose value is desired.
- *value*: Upon return, this parameter contains the string representation of the current value of the field specified in *field*. For fields that contain enum or set values, the string representation of the numeric value is returned, not the text description of this value. Date and UTC fields are not converted, so the Baan 3GL internal long values are returned as string. Normal numeric values are also returned as string (for example, -123.45), no conversion based on display formats or language dependent format is performed.
- *element*: Array element whose value is to be returned in the case of arrays or repeating fields.

Description

This returns the current value of a particular field from a specified running Infor ERP session. If the field is available in the field buffer, the value is taken from there. Otherwise, the value is extracted directly from the session. For information on the stpapi.* functions that cause the field buffer to be updated, see Chapter 4, “Special issues.”

Return Values

None.

Example

```
stpapi.put.field("dtfsa1101s000", "dtfsa100.seno", str$(i.seno))
retval = stpapi.find("dtfsa1101s000", error.msg)
if retval = 1 then
        stpapi.get.field("dtfsa1101s000", "dtfsa100.name", o.name)        stpapi.get.field("dtfsa1101s000",
"balance", o.balance)
endif
```

Explanation:

In the dtfsa1101s000 session, the record with key i.seno is searched. The values of the fields **dtfsa100.name**, which is the table field on the form, and **balance**, which is the calculated field on the form, are retrieved from the session.

Usage Notes

Function in dll created by creatdll:

```
Function extern domain <domain-name> <fs-name>.get.<field-descr>()
```

ERP Baan5.x and ERP Enterprise (LN) only: If the AFS calls a synchronized session (multi-occurrence/single-occurrence) the fields must be taken from the session in which the field resides. This implies that if a field must be fetched from the single-occurrence dialog box, a synchronize call must have been performed (see stpapi.synchronize.dialog()).

For segmented fields, the get.field function must be performed on the separated segments. A get function on the segmented field itself will not work.

Example

```
retval = stpapi.find("dtfsa2500m000", error.msg)
if retval = 1 then
        stpapi.get.field("dtfsa2500m000", "dtfsa200.segm.segment.1", “1”)
        stpapi.get.field("dtfsa2500m000", "dtfsa200.segm.segment.2", “2”)
endif
```

Set Field Value in session

Syntax

```
void stpapi.put.field(string session, string field, string value, [long
element])
```

Arguments

- *session*: Name of the session this command is executed on.
- *field*: Name of the field whose value is desired.
- *value*: The value of the field specified in *field* is set to the contents of this parameter. Any necessary type conversion is performed, however, Date and UTC fields must be placed in the internal Baan 3GL format. Enum fields must be put in the internal byte value. Examples: str$(date.num()), str$(-123.45), str$(etol(tcyesno.yes)).
- *element*: Array element whose value is to be set in the case of arrays or repeating fields.

Description

This sets the current value of a particular field in a specified running Infor ERP session. If the field is available in the field buffer, the value is placed in the field until the value is processed by the field loop in the session. For more information, refer to Chapter 4, “Special issues.” If the field is not available in the field buffer, the field value is directly sent to the session with the put.var function.

Note that no field sections in the script are run during the put.field call, therefore, no validation is performed. These sections are called when all fields are processed for an insert or update call.

Return Values

None.

Example

```
stpapi.put.field("dtfsa1201s000", "seno.f", str$(i.seno))
stpapi.put.field("dtfsa1201s000", "seno.t", str$(i.seno))
stpapi.put.field("dtfsa1201s000", "proc.date", str$(date.num()))
stpapi.put.field("dtfsa1201s000", "do.update", str$(etol(dtyesno.no)))
stpapi.continue.process("dtfsa1201s000", error.msg)
```

Explanation:

For a processing session, the input fields are sent to the session, and the continue process function of the session is run.

Usage Notes

Function in dll created by creatdll:

`Function extern void <fs-name>.put..<field-descr>(const domain <domain-name> value)`

The case and alignment of the passed string values is not relevant. The AFS automatically converts the value to the correct domain.

ERP Baan 5.x and ERP Enterprise (LN)) only: If fields are sent to a single-occurrence session, which is synchronized with a multi-occurrence session, the values must be put after a stpapi.synchronize.dialog() call is issued to the multi-occurrence session, because otherwise the single-occurrence session is activated without a link to the multi-occurrence session.

For segmented fields, the put.field function must be performed on the separated segments. A put function on the segmented field itself will not work.

Example

```
stpapi.put.field("dtfsa2500m000", "dtfsa200.segm.segment.1", “1”)
stpapi.put.field("dtfsa2500m000", "dtfsa200.segm.segment.2", “2”)
stpapi.put.field("dtfsa2500m000”, “dtfsa200.int”, “1”)
stpapi.insert("dtfsa2500m000", error.msg)
```

Clear All fields

Syntax

```
void stpapi.clear(string session)
```

Arguments

Session	Name of the session this command is executed on.

Description

This function fills in an ‘empty’ value (unset) for all input fields of the form.

Example

```
stpapi.clear(“dtfsa1101s000”)
stpapi.put.field("dtfsa1101s000", "dtfsa101.seno", str$(i.seno)) stpapi.put.field("dtfsa1101s000", "dtfsa101.name", name)

retval1 = stpapi.insert("dtfsa1101s000", true, error.msg)
if not retval1 then
        retval2 = stpapi.recover("dtfsa1101s000", recover.msg)
endif
```

Explanation:

Before you perform the put.field actions, you first empty all fields using the clear function.

Usage Notes

Function in dll created by creatdll:

```
Function extern void <fs-name>.clear()
```

Note:

In early versions of Functionserver, you required this stpapi.clear function to clear the put field buffers. The newer versions will, after an action such as insert automatically, reset the put field buffers.

Insert Record in session

Syntax

```
long stpapi.insert (string session, long do.save, ref string err.mesg)
```

Arguments

- *Session*: Name of the session on which this command is run.
- *do.save:* Flag that specifies whether a database commit must take place within this function. If *do.save* is set to 1, the update.db choice section of the session will be run before this function returns. If *do.save* is set to 0, update.db is not run. In this case, a stpapi.save() function must be called afterwards to update the database.
- *err.mesg* : This parameter contains the text of the error message if the function cannot complete normally.

Description

This inserts the current record of the specified session into the database. The values of the fields in the session must be set before calling this function.

Return Values

0 Record not inserted or save failed: *err.mesg* is filled with the reason.

1 Record inserted: *err.mesg* is empty

Example

```
stpapi.put.field("dtfsa1101s000", "dtfsa101.seno", str$(i.seno)) stpapi.put.field("dtfsa1101s000", "dtfsa101.name", name)

retval1 = stpapi.insert("dtfsa1101s000", true, error.msg)
if not retval1 then
        retval2 = stpapi.recover("dtfsa1101s000", recover.msg)
endif
```

Explanation:

The fields to be filled in the table are sent to the session and the insert function is called.

Usage Notes

Function in dll created by creatdll:

```
Function extern long <fs-name>.insert(long do.update, ref string error))
```

If *err.mesg* is filled, stpapi.recover() must be called before any other stpapi.* commands are issued to this session, or the record can be inserted into the database because the stpapi.end.session() call will perform an update.db action.

These error messages can occur:

- Session not available.
- Command disabled; no insert is possible in the current state of the session.
- Editable synchronized dialog box not started (ERP Baan 5.x and ERP Enterprise (LN) only).
- Any error message from the session.

Only use this function with ‘do.save’ 0 if the calling program must distinguish between errors raised by the insert (for example, check.inputs) and the save (for example, skip.io’s in before.write). Do not use it to buffer inserts, because inserts will not be buffered, because the next stpapi.insert() call will run an update.db for the previously inserted record.

If a record is inserted with a type 3 form, the function stpapi.change.view() must be called to set the correct key field values.

ERP Baan 5.x and ERP Enterprise (LN) only: If a record must be inserted with a synchronized single-occurrence dialog box, the dialog box must be synchronized before the first put.field function is called, because otherwise the field buffer is not present. The stpapi.insert() call must be issued to the multi-occurrence session, but the stpapi.put.field() calls to the single-occurrence session. For multi-occurrence sessions of type 3, make sure you call the stpapi.change.view() before the synchronization.

Update Record in session

Syntax

```
long stpapi.update(string session, long do.save, ref string err.mesg)
```

Arguments

- *session*: Name of the session this command is executed on.
- *do.save:* Flag that specifies whether a database commit must take place within this function. If *do.save* is set to 1, the update.db choice section of the session runs before this function returns. If *do.save* is set to 0, update.db does not run. In this case, a stpapi.save() function must be called afterwards to update the database.
- *err.mesg*: This parameter contains the text of the error message if the function cannot complete normally.

Description

This updates the current record of the specified session. The values of the fields in the session must be set before calling this function.

Return Values

0 Record not updated or save failed: *err.mesg* is filled with the reason

1 Record updated: *err.mesg* is empty

Example

```
stpapi.put.field("dtfsa1101s000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.find("dtfsa1101s000", error.msg)
if ret = 1 then
stpapi.put.field("dtfsa1101s000", "dtfsa101.name", name)
retval1 = stpapi.update("dtfsa1101s000", true, error.msg)
                if not retval1 then
                    retval2 = stpapi.recover("dtfsa1101s000", recover.msg)
     endif
endif
```

Explanation:

The record to be updated is searched in the session and its name field is updated.

USAGE NOTES

Function in dll created by creatdll:

```
Function extern long <fs-name>.update(long do.update, ref string error)
```

A record must be current in the session (for example, by a stpapi.find() call), otherwise, the results are unpredictable.

If *err.mesg* is filled, stpapi.recover() must be called before any other stpapi.* commands are issued to this session, otherwise, the record can be updated in the database, because the stpapi.end.session() call performs an update.db action.

These error messages can occur:

- Command disabled: No update is possible in the current state of the session.
- Editable synchronized dialog box not started (ERP Baan 5.x and ERP Enterprise (LN) only).
- Any error message from the session.

Only use this function with do.save 0 if the calling program must distinguish between errors raised by the update (for example, check.inputs) and the save, for example, skip.io's in before.rewrite. Do not use it to buffer updates, because updates will not be buffered since the next stpapi.find() call to make another record current will execute an update.db for the previously updated record.

ERP Baan 5.x and ERP Enterprise (LN) only: If a record must be updated with a synchronized single-occurrence dialog box, the dialog box must be synchronized before the first put.field function is called, otherwise, the field buffer is not present. The stpapi.update() call must be issued to the multi-occurrence session, but the stpapi.put.field() calls to the single-occurrence session.

Delete Record from session

Syntax

```
long stpapi.delete(string session, long do.save, ref string err.mesg)
```

Arguments

- *session*: Name of the session this command is executed on.
- *do.save:* This option must be true. Do.save = false is not supported for stpapi.delete. The option is in the function for compatibility reasons but the function will act as do.save = true.
- *err.mesg* : This parameter contains the text of the error message if the function cannot complete normally.

Description

This deletes the current record of the specified session in the database.

RETURN VALUES

0 Record not deleted or save failed: *err.mesg* is filled with the reason.

1 Record deleted: *err.mesg* is empty.

Example

```
stpapi.put.field("dtfsa1101s000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.find("dtfsa1101s000", error.msg)
if ret = 1 then
retval1 = stpapi.delete("dtfsa1101s000", true, error.msg)
                    if not retval1 then
                          retval2 = stpapi.recover("dtfsa1101s000", recover.msg)
      endif
endif
```

Explanation:

The record to be deleted is searched in the session and then deleted.

Usage Notes

Function in dll created by creatdll:

```
Function extern long <fs-name>.delete(long do.update, ref string error)
```

A record must be current in the session (for example, by a stpapi.find() call), otherwise, the results are unpredictable.

If *err.mesg* is filled, stpapi.recover() must be called before any other stpapi.* commands are issued to this session. Otherwise, the record can be deleted in the database, because the stpapi.end.session() call will perform an update.db action.

These error messages can occur:

- Command disabled: No update is possible in the current state of the session.
- Any error message from the session.

ERP Baan 5.x and ERP Enterprise (LN) only: If a multi-occurrence session is used with a synchronized dialog box, the delete action must be sent to the multi-occurrence session.

Save Session Updates to database

Syntax

```
long stpapi.save(string session, ref string err.mesg)
```

Arguments

- *session*: Name of the session on which this command is run.
- *err.mesg* : This parameter will contain the text of the error message if the function cannot complete normally.

Description

This runs the choice section update.db of the specified session. Note that the same effect can be achieved by issuing any of the following functions:

- stpapi.insert()
- stpapi.update()

  with the *do.save* parameter set to 1.

Return Values

0 Record not saved: *err.mesg* is filled with the reason.

1 Record saved: *err.mesg* is empty.

Example

```
stpapi.put.field("dtfsa1101s000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.find("dtfsa1101s000", error.msg)
if ret = 1 then
      stpapi.put.field("dtfsa1101s000", "dtfsa101.name", name)
                  retval1 = stpapi.update("dtfsa1101s000", false, error.msg)
                  if retval1 then
            retval2 = stpapi.save("dtfsa1101s000", false, error.msg)        endif
      if not retval1 or not retval2 then
               retval3 = stpapi.recover("dtfsa1101s000", recover.msg)
      endif
endif
```

Explanation:

The record to be updated is searched in the session and its name field is updated. The update function is called without running update.db directly. The record is saved to the database with the save function.

Usage Notes

Function in dll created by creatdll:

```
Function extern long <fs-name>.save(ref string error)
```

If *err.mesg* is filled, stpapi.recover() must be called before any other stpapi.* commands are issued to this session, otherwise, the record can be updated in the database as the stpapi.end.session() call will perform an update.db action.

These error messages can occur:

- Command disabled: No update.db is possible in the current state of the session.
- Any error message from the session.

ERP Baan 5.x and ERP Enterprise (LN) only: If a record must be saved with a synchronized single-occurrence dialog box, the stpapi.save() call must be issued to the single-occurrence session.

Use the stpapi.insert(), and stpapi.update() calls with the do.save flag set to True, as this is better for performance. Buffering inserts or updates and saving one time will not work, as subsequent stpapi.* function calls will call update.db implicitly. In case of errors, the calling program does not know whether the update.db of the previous record failed, or that the current record is rejected due to field values.

Recover Session updates

Syntax

```
long stpapi.recover(string session, ref string err.mesg)
```

Arguments

- *session*: Name of the session this command is executed on.
- *err.mesg*: This parameter will contain the text of the error message if the function cannot complete normally.

Description

This runs the choice section recover.set in the specified session.

Return Values

0 Record not recovered: *err.mesg* is filled with the reason.

1 Record recovered: *err.mesg* is empty.

Example

```
stpapi.put.field("dtfsa1101s000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.find("dtfsa1101s000", error.msg)
if ret = 1 then
      stpapi.put.field("dtfsa1101s000", "dtfsa101.name", name)
                  retval1 = stpapi.update("dtfsa1101s000", false, error.msg)
                  if retval1 then
            retval2 = stpapi.save("dtfsa1101s000", false, error.msg)
      endif
      if not retval1 or not retval2 then
               retval3 = stpapi.recover("dtfsa1101s000", recover.msg)
      endif
endif
```

Explanation:

The record to be updated is searched in the session and the record's name field is updated. The update function is called without running update.db directly. The record is saved to the database using the save function.

Usage Notes

Function in dll created by creatdll:

```
Function extern long <fs-name>.recover(ref string error)
```

If *err.mesg* is filled, you must call stpapi.end.session()before you issue any other commands to this session.

The error messages returned can be:

- Command disabled: No update is possible in the current state of the session.
- Any error message from the session.

ERP Baan 5.x and ERP Enterprise (LN) only: If a record must be recovered using a synchronized single-occurrence dialog box, the stpapi.recover() call must be issued to the single-occurrence session.

Use the stpapi.insert(), stpapi.update(), and stpapi.delete() calls with the do.save flag set to True, because this setting is more favorable to performance. Buffering inserts or updates and saving one time will provide problems in case records are rejected, because subsequent calls will call update.db implicitly. The calling program does not know then which records failed.

Set Current Record for session

Syntax

```
long stpapi.find(string session [, ref string err.mesg])
```

Arguments

- *session*: Name of the session on which this command runs.
- *err.mesg*: This parameter contains the text of the error message if the function cannot complete normally.

Description

This finds the record in the session that corresponds with the current values of the session's key fields and makes the record current. The key field values must be set before calling this function.

The function is similar to the def.find function in Baan Windows.

Return Values

0 No record found: Empty table or error occurred and *err.mesg* is filled.

1 Record found: *err.mesg* is empty.

2 A record different from the one requested was found: *err.mesg* is empty.

Example

```
stpapi.put.field("dtfsa1101s000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.find("dtfsa1101s000", error.msg)
if ret <> 1 then
      message("Record not found")
endif
```

Explanation:

The key field for the session is put and the record is made current.

Usage Notes

Function in dll created by creatdll:

```
Function extern long <fs-name>.find([string error(500)])
```

The behavior is the same as a find action through BW:

- If no record exists, no record is shown: Return value 0
- If the record is found, the record is selected: Return value 1

- If the record cannot be found, the next record is selected: Return value 2
- If no record exists, no records are shown: Return value 0

  The error messages returned can be:

- Any error message from the session given during the find action.

Mark Current Record for session

Syntax

```
long stpapi.mark(string session [, ref string err.mesg])
```

Arguments

- *session:* Name of the session this command is executed on.
- *err.mesg*: This parameter will contain the text of the error message if the function cannot complete normally.

Description

This marks the record in the session, which is made current by the stpapi.find() or one of the stpapi.browse.set() functions.

Return Values

0 Record is not marked: *err.mesg* is filled

1 Record marked: *err.mesg* is empty

Example

```
stpapi.put.field("dtfsa1501m000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.find("dtfsa1501m000", error.msg)
if ret <> 1 then
        message("Record not found")
else
        ret = stpapi.mark("dtfsa1501m000", error.msg)
        if ret then
                stpapi.form.command("dtfsa1501m000", 5, "do.something",
                                error.msg)
        endif
endif
```

Explanation:

Because the form command needs a record to be marked, the found record is marked before the form command is called.

Usage Notes

Function in dll created by creatdll:

```
Function extern long <fs-name>.mark([string error(500)])
```

A record must be current in the session, for example, by a stpapi.find() call, otherwise, the results are unpredictable.

These error messages can occur:

- Any error message from the session given during the mark action.

  Only one record can be marked.

Browse Session records

Syntax

```
long stpapi.browse.set(string session, string option [, ref string err.mesg])
```

Arguments

- *session*: Name of the session this command is executed on.
- *option*: This parameter indicates what type of browse action is required. The possible values are first.set, next.set, prev.set, and last.set that correspond with the browse actions first record, next record, previous record, and last record respectively.
- *err.mesg*: This parameter will contain the text of the error message if the function cannot complete normally.

Description

This performs the specified browse action against the indicated session. The resulting record is set as current.

Return Values

0 No record found:
Empty table, no next or previous record or error occurred and *err.mesg* is filled.

1 Record found: *err.mesg* is empty

Example

```
ret = stpapi.browse.set("dtfsa1101s000", "first.set", error.msg)
while ret
                stpapi.get.field("dtfsa1101s000", "dtfsa100.name", o.name)
    stpapi.get.field("dtfsa1101s000", "balance", o.balance)
    rprt_send()
```

```
    ret = stpapi.browse.set("dtfsa1101s000", "next.set", error.msg)
endwhile
```

Explanation:

This piece of code browses through all records of a session and prints the fields.

Usage Notes

Functions in dll created by creatdll:

```
Function extern long <fs-name>.first([string error(500)])
Function extern long <fs-name>.next([string error(500)])
Function extern long <fs-name>.previous([string error(500)])
Function extern long <fs-name>.last([string error(500)])
```

For next.set and prev.set a record must be current in the session (for example, by a stpapi.find() call), otherwise the results are unpredictable.

These error messages can occur:

- Any error message from the session given during the browse action.

Set Current View for session

Syntax

```
long stpapi.change.view(string session [, ref string err.mesg])
```

Arguments

- *session*: Name of the session on which this command is run.
- *err.mesg*: This parameter will contain the text of the error message if the function cannot complete normally.

Description

This sets the current view for sessions with forms of type 3 (multiple occurrence plus view). The field values of the view fields must be set prior to calling this function.

The function is similar to the def.find function in Baan Windows.

Return Values

0 Empty view found or error occurred and *err.mesg* is filled

1 View found: *err.mesg* is empty

2 Another view found: *err.mesg* is empty

Example

```
stpapi.put.field("dtfsa1501m000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.change.view("dtfsa1501m000", error.msg)
if not ret then
      message(error.msg)
endif
```

Explanation:

The **View** field in the session is changed to the given key.

Usage Notes

Functions in dll created by creatdll:

`Function extern long <fs-name>.set.view([string error(500)])`

The behavior is the same as a find/new group action through BW:

- If no records exist, no record is shown: Return value 0
- If at least one record in the view exists, the first record in that view is selected: Return value 1
- If the view cannot be set due to either authorizations or application logic, the first record in the next view is selected: Return value 2
- If no next group is found, no records are shown: Return value 0.

  The error messages returned can be:

- Any error message from the session given during the browse action.

  Before you can insert a record on a type 3 form, the view must have been changed to the desired key fields.

Browse Session Views

Syntax

```
long stpapi.browse.view(string session, string option [, ref string
err.mesg])
```

Arguments

- *session*: Name of the session this command is executed on.
- *option*: This parameter indicates what type of browse action is required. The possible values are:
  - first.view: Corresponds to the first view browse action.
  - next.view: Corresponds to the next view browse action.
  - prev.view: Corresponds to the previous view browse action.

  - last.view: Corresponds to the last view browse action.
- *err.mesg*: This parameter will contain the text of the error message if the function cannot complete normally.

Description

This runs the specified browse action against the indicated session. The first record of the view being read is set as Current.

Return Values

0 No view found or error occurred and *err.mesg* is filled

1 View found: *err.mesg* is empty

Example

```
stpapi.put.field("dtfsa1501m000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.change.view("dtfsa1501m000", error.msg)
if ret then
      ret = stpapi.browse.view("dtfsa1501m000", "prev.view", error.msg)
      if ret then
            message("View before " & str$(i.seno) & " found")
endif
```

Explanation:

A check is performed whether a view is present before a given view.

Usage Notes

Functions in dll created by creatdll:

```
Function extern long <fs-name>.first.view([string error(500)])
Function extern long <fs-name>.next.view([string error(500)])
Function extern long <fs-name>.previous.view([string error(500)])
Function extern long <fs-name>.last.view([string error(500)])
```

These error messages can occur:

- Any error message from the session given during the browse action.

Synchronize Multi-occurrence and Single-occurrence sessions

Syntax

```
long stpapi.synchronize.dialog(string session, string mode, ref string
err.mesg)
```

Arguments

- *session*: Name of the multi-occurrence session on which this command runs.
- *mode:* The mode in which the synchronized dialog box must be started. The possible values are:
  - Add: The dialog box starts and is synchronized in Edit mode. View fields are sent from the multi-occurrence session to the single-occurrence session. Use this mode before a stpapi.insert() call.
  - Modify: The dialog box starts and is synchronized in Edit mode. Use this mode before a stpapi.update() call.
  - Display: The dialog box starts in Display mode.
  - " ": No dialog box is synchronized. For future use when multi-occurrence and single-occurrence sessions have the same code, focus is set to the multi-occurrence session.
- *err.mesg*: This parameter will contain the text of the error message if the function cannot complete normally.

Description

This function synchronizes a multi-occurrence session with the session's registered synchronized dialog box. Depending on the mode, the session with the editable window is synchronized or the read-only window.

Return Values

0 Sessions could not be synchronized: *err.mesg* is filled.

1 Sessions are synchronized: *err.mesg* is empty.

Example

```
stpapi.put.field("dtfsa1501m000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.find("dtfsa1501m000", error.msg)
if ret then
      ret = stpapi.synchronize.dialog("dtfsa1501m000", "modify", error.msg)
      if ret then
            stpapi.put.field("dtfsa1101s000", "dtfsa101.name", new.name)
                        ret = stpapi.update("dtfsa1501m000", true, error.msg)
      endif
endif
```

Explanation:

The record is searched in the multi-occurrence session. When found, the synchronized dialog box is started, the field changed, and the record updated.

Usage Notes

Functions in dll created by creatdll:

```
Function extern long <fs-name>.synchronize.dialog(string mode, ref string
error)
```

This function is for ERP Baan 5.x and ERP Enterprise (LN) only.

The error messages returned can be:

- Command disabled; no insert/update is possible in the current state of the session.
- Session has no synchronized dialog box.

You must use this function to get the multi-occurrence session synchronized with the session's connected single-occurrence dialog box. You must call this function before an insert or update call. In addition, you must call this function before you run a Form command on the single-occurrence session, because the editable and display variants can have various sets of form commands. To retrieve data from the single-occurrence dialog box, you must also run the synchronization.

Send Start processing command to session

Syntax

```
void stpapi.continue.process(string session, ref string err.mesg)
```

Arguments

- *session*: Name of the session on which this command runs.
- *err.mesg*: This parameter contains the text of the error message if the function cannot complete normally.

Description

This causes the choice option cont.process to be executed in the specified session.

Return Values

None.

Example

```
stpapi.put.field("dtfsa1201s000", "seno.f", str$(i.seno))
stpapi.put.field("dtfsa1201s000", "seno.t", str$(i.seno))
stpapi.put.field("dtfsa1201s000", "proc.date", str$(date.num()))
stpapi.put.field("dtfsa1201s000", "do.update", str$(etol(dtyesno.no)))
stpapi.continue.process("dtfsa1201s000", error.msg)
```

Explanation:

For a processing session, the input fields are sent to the session, and the continue process function of the session runs.

Usage Notes

Functions in dll created by creatdll:

`Function extern void <fs-name>.continue(ref string error)`

This function is for ERP Baan IV only. For ERP Baan 5.x and ERP LN (6.x), use the stpapi.form.command().

The error messages returned can be:

- Any error message from the session.

When a report is printed, the report specifications must have been set by stpapi.set.report().

Set Session Report parameters

Syntax

```
void stpapi.set.report(string session, string reportname, string device, ref
string err.mesg)
```

Arguments

- *session*: Name of the session on which this command runs.
- *reportname:* Valid Infor ERP report code for desired report.
- *device:* Valid Infor ERP device code for desired device.
- *err.mesg*: This parameter contains the text of the error message if the function cannot complete normally.

Description

This selects the report to be printed and the device to be printed to if you call stpapi.print.report(), stpapi.continue.process(), or stpapi.form.command().

Return Values

None.

Example

```
stpapi.put.field("dtfsa1401m000", "seno.f", str$(i.seno))
stpapi.put.field("dtfsa1401m000", "seno.t", str$(i.seno))
stpapi.set.report("dtfsa1401m000", "rdtfsa140111000", pr.device, error.msg)
if isspace(error.msg) then
      stpapi.continue.process("dtfsa1401m000", error.msg)
endif
```

Explanation:

The specified session prints the given report.

Usage Notes

Functions in dll created by creatdll:

```
Function extern void <fs-name>.set.report(string reportname, string device,
ref string error)
```

The case and alignment of the passed string values is not relevant. The AFS automatically converts the value to the correct domain.

The error message that can be returned is for future use. Currently, no errors will be returned by the function. Errors about invalid reports or devices are returned by the subsequent stpapi.print.report(), stpapi.continue.process(), or stpapi.form.command() calls.

Some sessions determine the report to be printed based on the input on the form. If the report code is already known in the session when the report is opened, the value set in the AFS is ignored.

Only one report can be set. Therefore, sessions that print more than one report in one execution cannot be executed with the AFS.

To print the report to a file, use a write-file or rewrite file device. To print the data to another file then the default file for the device, you must also set the spool.fileout variable:

```
stpapi.put.field(“<sessioncode>”, “spool.fileout”, myfile)
```

Send Print command to session

Syntax

```
void stpapi.print.report(string session, ref string err.mesg)
```

Arguments

- *session*: Name of the session this command is executed on.
- *err.mesg*: This parameter will contain the text of the error message if the function cannot complete normally.

Description

This causes the choice option print.data to run in the specified session.

Return Values

None.

Example

```
stpapi.put.field("dtfsa1401m000", "seno.f", str$(i.seno))
stpapi.put.field("dtfsa1401m000", "seno.t", str$(i.seno))
stpapi.set.report("dtfsa1401m000", "rdtfsa140111000", pr.device, error.msg)
if isspace(error.msg) then
    stpapi.print.report("dtfsa1401m000", error.msg)
endif
```

Explanation:

For a print session, the input fields are sent to the session, and the print.data function of the session runs.

Usage Notes

Functions in dll created by creatdll:

`Function extern void <fs-name>.print(ref string error)`

This function is for ERP Baan IV only. For ERP Baan 5.x and ERP Enterprise (LN), use stpapi.form.command().

These error messages can occur:

- Report not found.
- Device not found.
- Any error message from the session.

The report specifications must have been set by stpapi.set.report().

End session

Syntax

```
void stpapi.end.session(string session [, ref string err.mesg)
```

Arguments

session: Name of the session on which this command runs.

Description

This ends the specified session.

Return Values

None.

Example

```
stpapi.put.field("dtfsa1101s000", "dtfsa101.seno", str$(i.seno)) stpapi.put.field("dtfsa1101s000", "dtfsa101.name", name)

retval1 = stpapi.insert("dtfsa1101s000", true, error.msg)
if not retval1 then
        retval2 = stpapi.recover("dtfsa1101s000", recover.msg)
endif
stpapi.end.session("dtfsa1101s000")
```

Explanation:

A record is inserted in the session and the session is ended.

Usage Notes

Functions in dll created by creatdll:

```
Function extern void <fs-name>.end([string error(500)])
```

This error message can be:

- Any error message from the session.

Although sessions are started implicitly (when used for the first time, see Chapter 4, “Special issues”), you must end sessions explicitly.

If the session has activated another session (see also stpapi.handle.subprocess()), you must first end the subsession, because the main session might be waiting until the subsession ends. In this case, the main session cannot accept the protocol message to end.

Execute session user option

Syntax

```
void stpapi.application.option(string session, long form, long option, ref
string err.mesg)
```

Arguments

- *session*: Name of the session on which this command runs.
- *Form:* Form number on which user option must be executed. User options, such as the commands on the **Special** menu, are defined for each form.
- *option:* The option number to be run. User options appear as choice.user.x options in the Baan 4GL code where x is a number that ranges from 0 to 9. The value of x that corresponds to the option that you want to activate is provided as the value of this parameter.
- *err.mesg :* This parameter contains the text of the error message if the function cannot complete normally.

Description

This runs the specified user option in the session.

Return Values

None.

Example

```
stpapi.put.field("dtfsa1501m000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.find("dtfsa1501m000", error.msg)
if ret = 1 then
    stpapi.application.option("dtfsa1501m000", 2, 3, error.msg)
endif
```

Explanation:

For the record found, choice user.3 is run on the second form.

Usage Notes

Functions in dll created by creatdll:

```
Function extern void <fs-name>.<option description>(ref string error)
```

This function is for ERP Baan IV only. For ERP Baan 5.x and ERP Enterprise (LN), use stpapi.form.command().

This error message can be:

- Any error message from the session.

Execute session zoom option

Syntax

```
void stpapi.zoom.option(string session, long form, string zoom.prog, ref
string err.mesg)
```

Arguments

- *Session*: Name of the session on which this command runs.
- *Form:* Form number on which zoom must be executed. You must specify this form number, because some sessions depend on a particular form being active when you run the zoom.
- *zoom.prog:* Infor ERP session for the zoom. This must be the same as the zoom session specified for the choice field of *form*. If the choice specifies a zoom to a menu, this session must be one of those listed on the menu.
- *err.mesg :* This parameter contains the text of the error message if the function cannot complete normally.

Description

This performs a zoom in the specified session to the program given in *zoom.prog*.

Return Values

None.

Example

```
stpapi.put.field("dtfsa1501m000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.find("dtfsa1501m000", error.msg)
if ret = 1 then
    stpapi.handle.subproc("dtfsa1501m000", "dtfsa1201m000", "add")
    stpapi.zoom.option("dtfsa1501m000", 2, "dtfsa1201m000", error.msg)
    stpapi.continue.process("dtfsa1201m000", error.msg)
endif
```

Explanation:

For the record found, the session dtfsa1201m000 is run, which is connected to the **Choice** field or a menu choice of the menu connected to the **Choice** field.

Usage Notes

This function is for ERP Baan IV only. For ERP Baan 5.x and ERP Enterprise (LN), use stpapi.form.command().

This error message can be:

- Any error message from the main session, which is set in before.choice of choice.zoom.

Execute session form command

Syntax

```
void stpapi.form.command(string session, long command.type, string
command.prog, ref string err.mesg)
```

Arguments

- *Session*: Name of the session on which this command runs.
- *command.type:* The type of form command to be run. The following values apply:
  - 2: Session
  - 5: Function
- *command.prog:* The code of the session or the name of the function to be run.
- *err.mesg*: This parameter contains the text of the error message if the function cannot complete normally.

Description

This function causes the specified form command to run in the specified session.

Return Values

None.

Example

```
stpapi.put.field("dtfsa1501m000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.find("dtfsa1501m000", error.msg)
if ret = 1 then
      ret = stpapi.mark("dtfsa1501m000", error.msg)
      if ret = 1 then
            stpapi.form.command("dtfsa1501m000", 5, "calculate.vat",
                        error.msg)
      endif
endif
```

Explanation:

For the record found, the function calculate.vat is carried out.

Usage Notes

Functions in dll created by creatdll:

```
Function extern void <fs-name>.<form-command-desc>(ref string error)
```

This function is for Baan 5.x and ERP LN (6.x) only. For ERP Baan IV, use stpapi.continue.process(), stpapi.application.option(), or stpapi.zoom.option().

The error messages returned can be:

- Form command not found in session.
- Any error message from the session.

Whether an error message is returned in the function call depends on the way in which the call is coded in the session. See Chapter 4, “Special issues,” for more information.

The form command is searched in the session. If the command is found in one of the forms, the command is run.

If a report is printed, the report specifications must have been set by stpapi.set.report().

If the form command is of type Session, a stpapi.handle.subproc() call must first be issued.

Specify actions for subsessions

Syntax

```
void stpapi.handle.subproc(string session, string sub.prog, string action)
```

Arguments

- *session*: Name of the session on which this command runs.
- *sub.prog:* Infor ERP session code to which the action specified applies. This must be the session code of a valid subsession of the specified session or the menu code of a menu that can be started by the specified session.
- *Action:* The action to be taken when the subsession starts or which menu choice to be activated when the menu is activated.

Actions for subsessions:

- kill: Child process is killed as soon as the process starts
- ignore: The child process is ignored, although the parent will wait until the child ends.
- send: All future stpapi.* function calls that use the name of the parent act on the child instead of the parent.
- add: The child is added to the list of sessions currently under control. The child can, therefore, be controlled independently by issuing stpapi.* function directly against the child’s session name

Menu choice:

- Menu choice of the session to be started converted to a string

Description

This sets the action that is taken when the specified subsession is invoked from the specified session or the menu choice to be activated when the menu is invoked. Note that a stpapi.handle.subproc() must also be called for the session activated by the menu choice.

Return Values

None.

Example

```
stpapi.put.field("dtfsa1501m000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.find("dtfsa1501m000", error.msg)
if ret = 1 then
      stpapi.handle.subproc("dtfsa1501m000", "dtfsa1201m000", "add")
      stpapi.zoom.option("dtfsa1501m000", 1, "dtfsa1201m000", error.msg)
      stpapi.put.field("dtfsa1201m000", "dtfsa101.date", str$(date.num()))
      stpapi.continue.process("dtfsa1201m000", error.msg)
      stpapi.end.session("dtfsa1201m000")
endif
stpapi.end.session("dtfsa1501m000")
```

Explanation:

For the record found, the session dtfsa1201m000 is run, which is connected to the **Choice** field or a menu choice of the menu connected to the **Choice** field. To also send messages to the subsession, the stpapi.handle.subproc() is called.

Usage Notes

Functions in dll created by creatdll:

`Function extern void <fs-name>.handle.sub.process(string sub.process, string action)`

If the subprocesses are a session without form, you cannot communicate with the session through the AFS. In these cases, you cannot define the action for the subsessions of this process.

For this situation, you can define the action for the subprocesses of that session in the group session, which is the session where the pid is the same as the gid of this subsession in the process list. In addition, the intermediate session must be defined with action Ignore.

This function must always be called when the session starts a subprocess called through the AFS when the subprocess expects user interaction, such as filling fields or pressing buttons.

Get Messages from session

Syntax

`string stpapi.get.mess.code(string session [, ref string err.mesg])`

Arguments

- *session*: Name of the session on which this command runs.
- *err.mesg:* Text of the message.

Description

This retrieves the messages generated by the indicated session as a result of an stpapi.* function call. The messages are returned in the opposite order as generated in the session. Messages generated by the AFS or 4GL-Engine are also returned. These messages are generated when the AFS is not used correctly by the programmer, for example, wrong order of function calls.

Return Values

String that contains code of error message or the empty string if no error message is found or only the error text is filled.

Example

```
stpapi.put.field("dtfsa1101s000", "dtfsa101.seno", str$(i.seno)) stpapi.put.field("dtfsa1101s000", "dtfsa101.name", name)

retval1 = stpapi.insert("dtfsa1101s000", true, error.msg)
if not retval1 then
    while true
        error.code = stpapi.get.mess.code("dtfsa1101s000", error.msg)
        if isspace(error.msg) then
            break
        endif
                                                    rep.message = error.msg
        rprt_send()
    endwhile
    retval2 = stpapi.recover("dtfsa1101s000", recover.msg)
endif
```

Explanation:

A record is inserted. If an error occurred, all errors raised by the session are retrieved and printed.

Usage Notes

Functions in dll created by creatdll:

```
Function extern string <fs-name>
.get.last.message.code([string error(500)])
```

For more information, refer to Chapter 4, “Special issues.”

Set answers to questions in session

Syntax

```
void stpapi.enum.answer(string session, string question, bset answer)
```

Arguments

- *session*: Name of the session on which this command runs.
- *question:* Infor ERP question code. This question code must be valid in the session, that is, the session must ask this question when the user interface is used.
- *answer:* The enum or set answer to be supplied as answer to the question. This must be expressed as the enum value as opposed to the numeric equivalent for example, tcyesno.yes i.s.o. 1.

Description

This sets the answers to questions that occur while the session runs.

Return Values

None.

Example

```
stpapi.put.field("dtfsa1101s000", "dtfsa101.seno", str$(i.seno))
stpapi.put.field("dtfsa1101s000", "dtfsa101.name", new.name)
stpapi.enum.answer("dtfsa1101s000", "dtfsa1101a", tcyesno.yes)
ret = stpapi.insert("dtfsa1101s000", true, error.msg)
```

Explanation:

The session prompts the user whether to continue if a record is already present with the same name. The default answer in the session is No, but you must continue.

Usage Notes

Functions in dll created by creatdll:

`Function extern long <fs-name>.define.enum.answer(string question, bset answer)`

This function must only be used for questions for which the default answer in the session must be overruled.

If the same question is used more than once in the same session, you can define only one answer.

Change Sort Order

Syntax

```
long stpapi.sort.by(string session, string sortorder, ref string err.mesg)
```

Arguments

- *session*: Name of the session this command is executed on.
- *sortorder*: The number of the sorting. This number is not the index number but the number in sequence as presented to the enduser.
- *err.mesg*: This parameter will contain the text of the error message if the function does not complete normally.

Description

This function will change the sorting order of the specified session.

Return Values

0 Record not recovered: err.mesg is filled with the reason.

1 Record recovered: err.mesg is empty.

Usage Notes

ERP Baan IV and ERP Baan 5.0c and ERP Enterprise (LN) only.

Chapter 4 Special issues

4

To start application sessions

The application session starts when the first AFS call for that session runs. However, a number of functions return an error message if they are the first function call for a specific session:

stpapi.insert()	One or more stpapi.put.field() calls must have been performed.
stpapi.update()	A record must be current, so a stpapi.find() must have been performed.
stpapi.delete()	A record must be current, so a stpapi.find() must have been performed.
stpapi.save()	A stpapi.insert(), stpapi.update() must have been performed.
stpapi.recover()	A stpapi.insert(), stpapi.update() must have been performed.
stpapi.continue.process()	The session parameters must have been filled.

Field buffer

Field loop

During insert and update calls and also continue.process, print.data, and form.command calls on type 4 forms, the field loop of the 4GL-Engine is executed.

All fields on all forms are processed. For these fields, the 4GL-sections are executed. Between the sections before.input and before.checks, the value is extracted from the field buffer when the value is filled by a stpapi.put.field() call. To find out whether the field is put, another array is used; this array contains an entry for each field, which indicates whether the field is put. This array is initialized after each stpapi.* function call, which caused the session to use the values (for example, stpapi.insert()).

Field buffer as input

The following AFS functions cause the 4GL-Engine to take the values from the field buffer:

Function	Remark
stpapi.insert()	
stpapi.update()	
stpapi.find()	Only key fields
stpapi.change.view()	Only view fields
stpapi.continue.process()	Only in a form of type 4, for other form types the record must be made current (with a find or browse function)
stpapi.print.report()	Only in a form of type 4, for other form types the record must be made current (with a find or browse function)
stpapi.application.option()	Only in a form of type 4, for other form types the record must be made current (with a find or browse function)
stpapi.zoom.option()	Only in a form of type 4, for other form types the record must be made current (with a find or browse function)
stpapi.form.command()	Only in a form of type 4, for other form types the record must be made current (with a find or browse function)

For the form fields, the field loop is run, as described in “Field loop,” in this chapter. Messages from the field checks, or messages raised during the processing, are returned.

Field buffer as output

The following AFS functions cause the 4GL-Engine to update the values in the field buffer:

Function	Remark
stpapi.insert()	Field buffer is updated with the values written to the database (for example, the determined order number overwrites the input value, which only contained a series number)
stpapi.update()	Field buffer is updated with the values written to the database (for example, the determined order number overwrites the input value, which only contained a series number)
stpapi.save()	Field buffer is updated with the values written to the database (for example, the determined order number overwrites the input value, which only contained a series number)
stpapi.recover()	Restores the old values
stpapi.find()	The field buffer is updated with the values of the record, which is made Current.

Function	Remark
stpapi.browse.set()	The field buffer is updated with the values of the record, which is made Current.
stpapi.browse.view()	By default, the first set of the view is returned, but the application session can react differently on the view events; the field buffer is updated with the values of the current record in the session
stpapi.synchronize.dialog()	The field buffer of the single-occurrence session is updated (only of Modify and Display) with the values of the current record of the single-occurrence session

Message handling

Introduction

One of the main issues of using the AFS is modifying the Infor ERP database through the business logic of the sessions. Validations performed in the session scripts must also be performed when AFS runs the session.

In case of incorrect data, error messages are returned through the AFS to the calling program. This section describes this message handling.

Functions

The following Baan 4GL functions can raise messages:

- void message(string mess_str(.) [, arg, ...])
- void mess(string messcode(14), long mode [, arg, ...])
- void set.input.error(string messcode(14) [, arg, ...])
- void dal.set.error.message(string mess.or.code [, arg ...])[2]
- void skip.io(string mesg(14) [, ...])
- void abort.io(string mesg(14) [, ...])

All messages given by these functions are returned by the AFS; however, messages raised by mess() or message() are treated as warning, except when followed by set.input.error(), skip.io(), abort.io(), or choice.again().

[2] Baan 5.x and 6.1 only

If the functions set.input.error(), skip.io(), and abort.io() are called with an empty argument and without a mess() or message() call before, an error message is generated. The same applies to the input.again() function.

If a choice.again() is called without a mess() or message() call before it, a Command cancelled warning is generated.

An error is returned in the argument of most stpapi.* functions, warnings can only be retrieved by the function stpapi.get.mess.code().

Message array

Messages raised by the session are kept in an array, with a maximum of 20 messages. If more than 20 messages are raised, the twenty-first message will be: “More than 20 messages raised by the session, other messages are lost.”

The function stpapi.get.mess.code() gives access to this array. The array has two fields: error code and error text.

The error code can be empty for messages raised by the function message(). In the first call to this function, the last message is returned, then the previous one, etc..

Example of message array:

Code	Text
dtfsas0002	Price must be filled
dtfsas0001	Note: Insufficient inventory

In this example, the session first raised a warning, in the check.input of the quantity field, with only a mess() function. In the check.input of the **Price** field set.input.error(dtfsas0002) was called, or mess(dtfsas0002, 1) followed by set.input.error(“”). The string “Price must be filled” is also returned in the error argument of the stpapi.insert() call.

If the message array only contains the warning dtfsas0001, no string is returned in the error argument of the stpapi.insert().

For information on how to handle the message array, refer to the example code in Chapter 3, "Baan 4GL engine primitives.”

Before each stpapi.* call, the array is cleaned up, except stpapi.put.field(), stpapi.get.field() and stpapi.get.mess.code().

Generated error messages

In some situations an error can occur while no error message appears:

- set.input.error, skip.io or abort.io are called without a message code and no message or mess function is called before. For skip.io and abort.io, a default message exists in the 4GL-Engine;

this one is returned. For set.input.error, no default error message exists, and the error Input cancelled on field …… is returned.

- Input.again is called without a preceding message or mess function call, and the error Input cancelled on field …… is returned.
- Choice.again is called without an error message set before. As the AFS does not know whether this function is called before or after the processing, no error message is generated. A warning Command cancelled is written to the message array.

Consider the following examples:

```
choice.cont.process:                    choice.cont.process:
on.choice:                              on.choice:
processing()                            if some.condition then
                                  choice.again()                choice.again()
                                        else
                                              processing()
                                         endif
```

In the first example, processing is performed, while in the second no processing is performed. The AFS cannot see the difference. In both situations, the warning Command cancelled is written to the message array and the error message argument of the stpapi.continue.process() or stpapi.form.command() remains empty.

Again Choice.again()

When a choice.again() function is used in the application session, and a message is raised before, the error message argument of stpapi.continue.process() and stpapi.form.command() is filled. However, this will not always imply that an actual error occurred.

Consider the following example:

```
choice.cont.process:
on.choice:
   processing()
   message(“Ready”)
   choice.again()
```

In this case, the error message argument is filled with Ready.

To handle this correctly in the calling programs, the programmer must know what happens in the session. If you cannot find out whether the session is run as desired, you must change the application session. Refer to Chapter 5, "Guidelines for Baan 4GL application sessions,” for more details.

Form commands

Form commands of type Function can also raise messages. Usually, a message is given and a return statement ends the function. In this case, the AFS treats this as warnings, and the stpapi.form.command() call does not return the error message directly. If a choice.again() ends the function, the stpapi.form.command() call returns the error message directly.

Messages from AFS and 4GL-Engine

The AFS itself can give several messages, for example, if functions are called in the incorrect order. Message codes are not assigned to all of these messages, therefore, the returned string of the function stpapi.get.mess.code() is empty. Because these messages can change, you must no longer parse these strings.

Multi-occurrence/Single-occurrence

ERP Baan 5.x and 6.1 only: For most inserts and updates in Infor ERP, a combination of a multi-occurrence and single-occurrence sessions are used. The programmer is responsible to get the sessions in a synchronized state. This is not done automatically for better performance and because the AFS cannot determine whether the editable dialog box or the read-only dialog box must be opened.

With the normal interface, synchronizing the multi-occurrence and single-occurrence session is an asynchronous process, so the processes do not wait until the other is synchronized. The AFS, however, waits until the synchronization is ready.

The sequence of the stpapi.* calls is as many of the same actions performed with the user interface as possible to carry out actions on a combination of a multi-occurrence and single-occurrence session.

Synchronization is necessary to:

- Insert records.
- Update records.
- Run a form command of the single-occurrence session.
- Retrieve data from the single-occurrence session.

The following sections describe each of these actions in more detail and provide examples for each action.

To insert records

To insert a record, you must start both the multi-occurrence session must be started and also the editable single-occurrence dialog box. The values for the new record must be sent to the single-occurrence session. The stpapi.insert() call is sent to the multi-occurrence session, the stpapi.recover() is sent to the single-occurrence session.

To insert records in a type 3 multi-occurrence session (with view), the view must first be set.

Example type 2, multi-occurrence:

```
ret = stpapi.synchronize.dialog("dtfsa1501m000", "add", error.msg)
if ret then
    stpapi.put.field("dtfsa1101s000", "dtfsa101.seno", str$(new.seno)
    stpapi.put.field("dtfsa1101s000", "dtfsa101.name", new.name)
    ret = stpapi.insert("dtfsa1501m000", true, error.msg)
```

```
    if not ret then
        ret = stpapi.recover("dtfsa1101s000", error.msg)
    endif
endif
stpapi.end.session("dtfsa1501m000", error.msg)
```

Explanation:

By calling the stpapi.synchronize.dialog() function, the multi-occurrence and single-occurrence sessions are both started. The fields are filled for the single-occurrence session. The insert is sent to the multi-occurrence session (the **Add** button is there). If the insert fails, the recover must be performed on the single-occurrence session (where the **Revert** button resides). The synchronized dialog box is killed when the multi-occurrence session is ended.

Example type 3, multi-occurrence:

```
stpapi.put.field("dtfsa1502m000", "dtfsa102.seno", str$(i.seno)
ret = stpapi.change.view("dtfsa1502m000", error.msg)
if ret = 1 then
    ret = stpapi.synchronize.dialog("dtfsa1502m000", "add", error.msg)
    if ret then
        stpapi.put.field("dtfsa1102s000", "dtfsa102.pono", str$(new.pono)
        stpapi.put.field("dtfsa1102s000", "dtfsa102.name", new.name)
        ret = stpapi.insert("dtfsa1502m000", true, error.msg)
        if not ret then
            ret = stpapi.recover("dtfsa1102s000", error.msg)
    endif
endif
stpapi.end.session("dtfsa1502m000", error.msg)
```

Explanation:

Similar to the previous example, however, a stpapi.change.view() must be called to fill the **View** field. During the synchronization, the view fields are copied to the single-occurrence session.

To update records

To update a record, the multi-occurrence session must be started and also the editable single-occurrence dialog box. The values for the new record must be sent to the single-occurrence session. The stpapi.update() call is sent to the multi-occurrence session, and the stpapi.recover() is sent to the single-occurrence session.

No difference exists between type 2 and type 3 multi-occurrence sessions, in both situations the record must be searched in the multi-occurrence session.

Example

```
stpapi.put.field("dtfsa1501m000", "dtfsa101.seno", str$(i.seno)
ret = stpapi.find("dtfsa1501m000", error.msg)
if ret = 1 then
    ret = stpapi.synchronize.dialog("dtfsa1501m000", "modify", error.msg)
    if ret then
```

```
        stpapi.put.field("dtfsa1101s000", "dtfsa101.name", new.name)
        ret = stpapi.update("dtfsa1501m000", true, error.msg)
        if not ret then
            ret = stpapi.recover("dtfsa1101s000", error.msg)
    endif
endif
stpapi.end.session("dtfsa1501m000", error.msg)
```

Explanation

The record to be modified is searched in the multi-occurrence session. The single-occurrence session is synchronized and the field is updated.

To run form commands

To run form commands of a synchronized single-occurrence session, the dialog box must also be synchronized if the correct record is found in the multi-occurrence session. The mode of the synchronized dialog box depends on the availability of the command in edit- and/or read-only mode.

Note that for form commands on the multi-occurrence session, no special action is necessary.

Example

```
stpapi.put.field("dtfsa1501m000", "dtfsa101.seno", str$(i.seno)
ret = stpapi.find("dtfsa1501m000", error.msg)
if ret = 1 then
    ret = stpapi.synchronize.dialog("dtfsa1501m000", "modify", error.msg)
    if ret then
        stpapi.form.command("dtfsa1101s000", 5, "some.function", error.msg)
    endif
endif
stpapi.end.session("dtfsa1501m000", error.msg)
```

Explanation

The record for which the form command must be executed is searched in the multi-occurrence session. The single-occurrence session is synchronized and the form command is executed.

To retrieve data from a synchronized dialog box

Required data that is not present in the multi-occurrence session must be retrieved from the synchronized dialog box.

Example

```
stpapi.put.field("dtfsa1501m000", "dtfsa101.seno", str$(i.seno))
ret = stpapi.find("dtfsa1501m000", error.msg)
if ret = 1 then
    ret = stpapi.synchronize.dialog("dtfsa1501m000", "display", error.msg)
    if ret then
```

```
        stpapi.get.field("dtfsa1101s000", 5, "balance", balance)
    endif
endif
stpapi.end.session("dtfsa1501m000", error.msg)
```

Explanation

The record to be modified is searched in the multi-occurrence session. The single-occurrence session is synchronized and the form command is executed.

Multi-Main table sessions

MMT sessions are not supported by Application Function Server. However since ES8.7 Solution 1016219 there is support for MMT sessions in Application Function Server Available.

Example

```
stpapi.put.field("tebmemmtcontr", "tebme004.index", “String1”)
ret = stpapi.find(("tebmemmtcontr", error.msg)
ret = stpapi.browse.set("tebme0113m000", "first.set", error.msg)
stpapi.get.field(("tebme0113m000", "tebme013.string", value)
stpapi.put.field(("tebmemmtcontr", "tebme004.index", "String2")
ret = stpapi.find(("tebmemmtcontr", error.msg)
ret = stpapi.browse.set("tebme0113m000", "first.set", error.msg)
stpapi.get.field(("tebme0113m000", "tebme013.string", value)
stpapi.end.session("tebmemmtcontr",  error.msg)
```

Explanation

The controller is started by the first stpapi.put.field on the controller (tebmemmtcontr). The satellites are not started.
The satellite session is started by the first command on the satellite session (tebme0113m000). The satellite session is started as a satellite of the just started controller.

To get the data from the satellite the Field Buffers need to be filled. See the topic Field Buffer in this chapter. In this case they are filled by the command stpapi.browse.set.

When the controller moved to another record (In the example done by a stpapi.find on the controller) the record buffers in the satellite are not updated. To get the new record buffers, again a command must be executed as described in the topic Field Buffers.

Debugging

When you test programs that use the AFS, errors can be in various places:

- The AFS is used incorrectly.
- The used Baan 4GL application session is not well suited for use by the AFS.

- An error can exist in the communication between the AFS and the 4GL-Engine.

Normal debugging of the AFS-using script and/or the Baan 4GL application session is not always possible, or does not give the appropriate information to solve the problem. For this reason, extra logging facilities are present to help the developers to solve problems.

This logging can be activated by setting the environment variable AFSLOG to a non-blank value:

- For ba: ba6.1 -set AFSLOG=1 or AFSLOG=1 ba6.1
- For bw: -set AFSLOG=1 in the **Command** field

During the execution of the AFS-using program, the results of the communication between the AFS and the 4GL-Engine are written to a log file afs.log in the user's home directory.

Example of logging:

```
LOGGING STARTED
2001-07-17
15->get.fields
15<-get.fields...
16->get.fields
16<-get.fields...
15->syncadd
15<-syncadd^A0
>dtfsa0101s000 put.field:dtfsa001.seno 3334
>dtfsa0101s000 put.field:dtfsa001.name name
>dtfsa0101s000 put.field:dtfsa001.date 730630
>dtfsa0101s000 put.field:dtfsa001.yeno 1
>dtfsa0101s000 put.field:dtfsa001.mult 3334
>dtfsa0101s000 enum.answer dtfsa0001:2
16->enum.answer^Adtfsa0001^A2
16<-enum.answer^A0
>dtfsa0501m000 insert
16->add.set+save
16<-add.set+save^A0^A0
<dtfsa0501m000
>dtfsa0501m000 end.session
15->end.program
15<-end.program^A0
```

In this logging, you can see the AFS functions, which the calling program calls, how the functions are translated to the internal protocol between the AFS and the 4GL-Engine, and what the 4GL-Engine returns.

Text management

The AFS does not support text management.

Chapter 5 Guidelines for Baan 4GL application sessions

5

This chapter provides several guidelines for Baan 4GL programmers, such as which constructions cannot be used in the program scripts, and when the session must be run through the AFS.

Predefined variable api.mode

In 4GL scripts, the predefined variable api.mode is present to know whether the session is started by the AFS. If different behavior is required for sessions that run in API-mode and sessions with a normal user-interface, this variable must be used.

If the variable api.mode is used in a DLL, this variable must be declared in the dll as extern long, otherwise, this can cause compile errors.

If you start a subsession from the active session, this activated session also starts automatically in api.mode.

Messages

As described in Chapter 4, “Special issues,” the message handling is very important. Because no user interface is present, nobody can see what happened exactly in the session. Functions such as set.input.error(), choice.again(), etc., which cause the session to cancel the requested action, can only be used if a message is given, or must not be called in api.mode.

Examples

```
choice.cont.process:

before.choice:

   if reprint = tcyesno.no and delete = tcyesno.no then

      choice.again()

         endif
```

In this example, the AFS never knows that nothing is done, while a normal user can see that the hourglass disappears immediately. To solve this, a message must be given, at least in api.mode:

```
choice.cont.process:
before.choice:
   if reprint = tcyesno.no and delete = tcyesno.no then
       if api.mode then
            mess("dtfsas0001", 1)
            |* No processing option selected
       endif
      choice.again()
         endif
```

An exception applies for choice.again(). Occasionally you must bypass the behavior of a session:

```
choice.zoom:
before.choice:
   if dtfsa001.type = dttype.normal then
       zoom.to$("dtfsa1500m000", z.session, "", "", 0)
       choice.again()
         endif
```

In this situation, the message Command cancelled is generated, and as a result, the AFS-user must ignore the message Command cancelled.

In some cases, messages are returned to the AFS that contain information only. If this message is followed by a choice.again(), the AFS treats this as an error:

```
choice.cont.process:
on.choice:
   do.all.processing()
    mess("dtfsas0002", 1)
   choice.again()
```

Preferably, use the following code:

```
choice.cont.process:
on.choice:
   do.all.processing()
    if not api.mode then
        mess("dtfsas0002", 1)
    endif
   choice.again()
```

Choice.again()

Only use choice.again() to stop the current choice section. Choice.again() used in field sections stops the field loop if records are inserted or updated by the AFS, so fields following on the same form and subsequent forms are not filled with the values from the field buffer.

Execute(…)

To help the end users, a choice sometimes starts automatically, for example, starting the add.set when no records are present. If the session runs in api.mode, starting choices automatically can result in undesired results.

Example

```
form.1:
init.form:
   execute(find.data)
    if filled.occ = = then
        execute(add.set)
    endif
```

Preferably, use the following code:

```
form.1:
init.form:
   if not api.mode then
      execute(find.data)
        if filled.occ = = then
            execute(add.set)
        endif
    endif
```

Hidden functionality

The AFS cannot reach functionality that is only accessible through zooming on field level. For example, if you click a zoom button for a field, a menu appears to maintain or display records. If no separate main session exists to maintain these records, no new record can be entered through the AFS.

Commands

If the session has more than one form, keep the form commands (standard and form specific) the same. AFS does not have a notion of current form.

Appendix A AFS-DLL example

This example is the AFS-DLL for the Maintain Areas (tcmcs0145m000) session in Infor ERP.

```
|-----------------------------------------------------------------------------
| File created by ttstpcreatdll, Date: 28/05/03
| Created for session: tcmcs0145m000
|-----------------------------------------------------------------------------

#pragma used dll ottstpapihand

function extern void f0145m000.put.area( const domain tccreg value )
{
DLLUSAGE
    Function to set area ( tcmcs045.creg ) in session tcmcs0145m000
    arg: - value to put in area
ENDDLLUSAGE
    stpapi.put.field( "tcmcs0145m000", "tcmcs045.creg", value )
}

function extern domain tccreg f0145m000.get.area( )
{
DLLUSAGE
    Function to get area ( tcmcs045.creg ) from session tcmcs0145m000
    return: - value of area
ENDDLLUSAGE
    string value(3)
    stpapi.get.field( "tcmcs0145m000", "tcmcs045.creg", value )
    return( value )
}

function extern void f0145m000.put.description( const domain tcdsca value )
{
DLLUSAGE
```

```
    Function to set description ( tcmcs045.dsca ) in session tcmcs0145m000
    arg: - value to put in description
ENDDLLUSAGE
    stpapi.put.field( "tcmcs0145m000", "tcmcs045.dsca", value )
}

function extern domain tcdsca f0145m000.get.description( )
{
DLLUSAGE
    Function to get description ( tcmcs045.dsca ) from session tcmcs0145m000
    return: - value of description
ENDDLLUSAGE
    string value(30)  mb
    stpapi.get.field( "tcmcs0145m000", "tcmcs045.dsca", value )
    return( value )
}

function extern void f0145m000.end([string error(500)])
{
DLLUSAGE
    Function to end connection to session tcmcs0145m000
ENDDLLUSAGE
    if get.argc() = 0 then
        stpapi.end.session( "tcmcs0145m000" )
    else
        error = get.string.arg(1)
        stpapi.end.session( "tcmcs0145m000" , error  )
        put.string.arg(1, error)
    endif
}

function extern long f0145m000.insert( long do.update, ref string error )
{
DLLUSAGE
    Function to insert a record in session tcmcs0145m000
    Fields must be put before calling this function
ENDDLLUSAGE
    return( stpapi.insert( "tcmcs0145m000", do.update, error ) )
}
```

```
function extern long f0145m000.update( long do.update, ref string error )
{
DLLUSAGE
    Function to update a record in session tcmcs0145m000
    Record must be made current and fields to be changed before calling
    this function
ENDDLLUSAGE
    return( stpapi.update( "tcmcs0145m000", do.update, error ) )
}

function extern long f0145m000.delete( long do.update, ref string error )
{
DLLUSAGE
    Function to delete a record in session tcmcs0145m000
    Record must be made current before calling this function
ENDDLLUSAGE
    return( stpapi.delete( "tcmcs0145m000", do.update, error ) )
}

function extern long f0145m000.mark([string error (500)])
{
DLLUSAGE
    Function to mark the current record in session tcmcs0145m000
ENDDLLUSAGE
    long ret
    if get.argc() = 0 then
        return( stpapi.mark( "tcmcs0145m000" ) )
    else
        error = get.string.arg(1)
        ret =  stpapi.mark( "tcmcs0145m000" , error  )
        put.string.arg(1, error)
        return( ret )
    endif
}

function extern long f0145m000.recover( ref string error )
{
DLLUSAGE
    Function to undo an update/insert/delete in session tcmcs0145m000
ENDDLLUSAGE
```

```
    return( stpapi.recover( "tcmcs0145m000", error ) )
}

function extern long f0145m000.save( ref string error )
{
DLLUSAGE
    Function to save an update/insert/delete in session tcmcs0145m000
ENDDLLUSAGE
    return( stpapi.save( "tcmcs0145m000", error ) )
}

function extern long f0145m000.find( [string error(500)] )
{
DLLUSAGE
    Function to find a record in session tcmcs0145m000
    Search fields must be put before calling this function
ENDDLLUSAGE
    long ret
    if get.argc() = 0 then
        return( stpapi.find( "tcmcs0145m000"  ) )
    else
        error = get.string.arg(1)
        ret =  stpapi.find( "tcmcs0145m000" , error  )
        put.string.arg(1, error)
        return( ret )
    endif
}

function extern long f0145m000.synchronize.dialog( string mode(8), ref string err.mesg)
{
DLLUSAGE
    Function to set up a synchronized dialog between multi-occurrence
    and single-occurrence session
ENDDLLUSAGE
    return( stpapi.synchronize.dialog("tcmcs0145m000", mode , err.mesg) )
}

function extern void f0145m000.clear( )
{
DLLUSAGE
```

```
    Function to clear the current record in session tcmcs0145m000
ENDDLLUSAGE
    stpapi.clear( "tcmcs0145m000" )
}

function extern void f0145m000.print( ref string error )
{
DLLUSAGE
    Function to start the print option in session tcmcs0145m000
ENDDLLUSAGE
    stpapi.print.report( "tcmcs0145m000", error )
}

function extern void f0145m000.set.report( const string reportname, const string device, ref string error )
{
DLLUSAGE
    Function to set report name and device for a subsequent print action
    in session tcmcs0145m000
ENDDLLUSAGE
    stpapi.set.report( "tcmcs0145m000", reportname, device, error )
}

function extern long f0145m000.first([string error(500)] )
{
DLLUSAGE
    Function to find the first record in session tcmcs0145m000
ENDDLLUSAGE
    long ret
    if get.argc() = 0 then
        return( stpapi.browse.set( "tcmcs0145m000", "first.set" ) )
    else
        error = get.string.arg(1)
        ret = stpapi.browse.set( "tcmcs0145m000", "first.set" , error )
        put.string.arg(1, error)
        return( ret )
    endif
}

function extern long f0145m000.next([string error(500)] )
{
```

```
DLLUSAGE
    Function to find the next record in session tcmcs0145m000
ENDDLLUSAGE
    long ret
    if get.argc() = 0 then
        return( stpapi.browse.set( "tcmcs0145m000", "next.set" ) )
    else
        error = get.string.arg(1)
        ret = stpapi.browse.set( "tcmcs0145m000", "next.set" , error )
        put.string.arg(1, error)
        return( ret )
    endif
}

function extern long f0145m000.previous([string error(500)] )
{
DLLUSAGE
    Function to find the previous record in session tcmcs0145m000
ENDDLLUSAGE
    long ret
    if get.argc() = 0 then
        return( stpapi.browse.set( "tcmcs0145m000", "prev.set" ) )
    else
        error = get.string.arg(1)
        ret = stpapi.browse.set( "tcmcs0145m000", "prev.set" , error )
        put.string.arg(1, error)
        return( ret )
    endif
}

function extern long f0145m000.last([string error(500)] )
{
DLLUSAGE
    Function to find the last record in session tcmcs0145m000
ENDDLLUSAGE
    long ret
    if get.argc() = 0 then
        return( stpapi.browse.set( "tcmcs0145m000", "last.set" ) )
    else
        error = get.string.arg(1)
```

```
        ret = stpapi.browse.set( "tcmcs0145m000", "last.set" , error )
        put.string.arg(1, error)
        return( ret )
    endif
}

function extern long f0145m000.set.view( [string error(500)] )
{
DLLUSAGE
    Function to define another view in session tcmcs0145m000
ENDDLLUSAGE
    long ret
    if get.argc() = 0 then
        return( stpapi.change.view( "tcmcs0145m000" ) )
    else
        error = get.string.arg(1)
        ret = stpapi.change.view( "tcmcs0145m000",  error )
        put.string.arg(1, error)
        return( ret )
    endif
}

function extern string f0145m000.get.last.message.code([string error(500)])
{
DLLUSAGE
    Function to get the code of the message which occurred on
    the last insert/update/delete/save/recover action in session tcmcs0145m000
ENDDLLUSAGE
    string ret(20)
    if get.argc() = 0 then
        return( stpapi.get.mess.code( "tcmcs0145m000" ) )
    else
        error = get.string.arg(1)
        ret = stpapi.get.mess.code( "tcmcs0145m000", error )
        put.string.arg(1, error)
        return( ret )
    endif
}

function extern string f0145m000.get.last.error( )
```

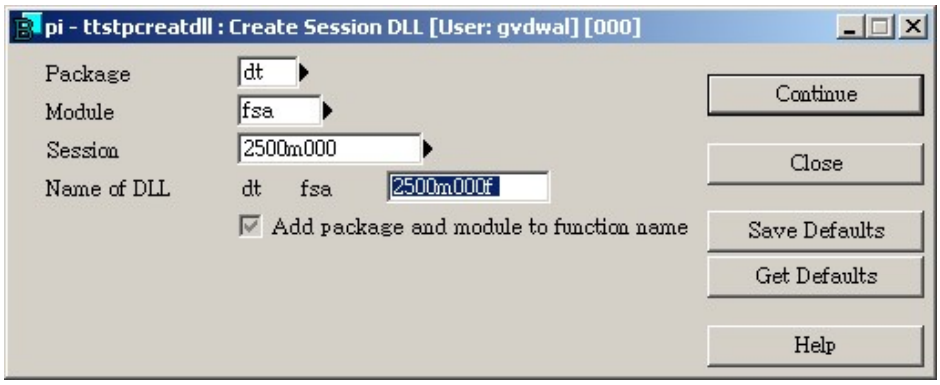
```
{
DLLUSAGE
    Function to get the error message which occurred on
    the last action in the Function Server
ENDDLLUSAGE
    return( stpapi.get.error( ) )
}

function extern void f0145m000.handle.sub.process( const string sub.process, const string action )
{
DLLUSAGE
    Function to define an action when a sub process is started.
    Possible actions are: add/send/ignore/kill
        add     - add child session to internal structure,
                session dll of child can be used
        send    - send all api calls to child instead of parent
        ignore  - child process is ignored, parent will wait
                until child exits (for background processes)
        kill    - child process is killed immediately
ENDDLLUSAGE
    stpapi.handle.subproc( "tcmcs0145m000", sub.process, action )
}

function extern void f0145m000.define.enum.answer( const string question, bset answer )
{
DLLUSAGE
    Function to define an answer on a question, when the default answer should not be taken.
ENDDLLUSAGE
    stpapi.enum.answer( "tcmcs0145m000", question, answer )
}
```

Appendix B StpCreatdll

This appendix illustrates the use of the Create Session DLL (ttstpcreatdll) session, which represents a tool for making a session DLL from a script of AFS functions calls.



The first three input fields, **Package**, **Module**, and **Session,** specify the session for which the Function Server DLL must be generated. Based on this information, a default name for the DLL is generated. This name is the full session code followed by an **f**. If you selected the **Add Package and Module to Function Name** check box, the DLL name includes the package and module of the session, for example, dtfsa2500m000. If you did not select this check box, the name does not include the names of the package and the module name, for example, 2500m000.

The generated functions of the session DLL always start with the name of this dll. Normally, a function must always start with a letter. However, if the DLL name does not start with a letter, this general convention is overruled and a function will start with a number.

After you generate a session DLL, you can find this DLL in the Maintain Scripts/Libraries session.