TEMENOS™

TEMENOS T24

Application Development

User Guide

Table of Contents

Overview

This document explains how to develop applications in T24.

T24 has an extensive infrastructure in place that enables the rapid development of business components. The creation and maintenance of T24 applications is based on a series of templates, which have had numerous incarnations throughout the life of T24. Now in its fourth iteration, this document details how applications are created in a step by step process. For an understanding of the genesis of the templates, refer to the appendix “Older Templates”. This document does not explain the process of the previous release of the templates – refer to “Template Programming V3”.

T24 is based on a one to one relationship between the data fields on the screen and the data fields in a file. The only way to enter data into T24 is via an APPLICATION which records the data entered and stores it in an associated file, field by field.

There are the following stereotypes:

- **Application** allows the full functionality of T24: data entry, authorisation, deletion, history restore, etc. e.g. CUSTOMER
- **Display** is used to simply view the data on a file maintained by the system rather than the user e.g. STMT.ENTRY
- **Utility** allows data entry and a Verification function which initiates a process, e.g. ENQUIRY.REPORT which records the selection criteria for an enquiry and then on Verification builds a report and spools it.

Creating an Application

Naming

The name of the application must be meaningful, should give some indication as to its purpose and should be prefixed by the product code of the application, where the product code should exist on EB.PRODUCT e.g. CR.CAMPAIGN.DEFINITION, PW.ACTIVITY, etc.

Note: The file name cannot exceed 40 characters, to allow for the addition of the company mnemonic and the $NAU or $HIS suffix.

Step 1 - Defining the Application

Having decided on the product and name of the application, the first step is to define the high level properties of the application. The example “TEMPLATE”, which ships with each release of T24, holds the properties for the application. Whilst there is tooling support for the creation of new applications, this section explains the mechanics behind the GUI.

Property	Example	Explanation
name	CR.CAMPAIGN.OPPORTUNITY	Full application name including the product prefix. Must be the same as the application name
title	Campaign Opportunities	Screen title
stereotype	H	See “Stereotypes”
product	CR	The product of the application. Must be an entry on EB.PRODUCT
subProduct	CAMPAIGN	The sub product of the application. Must be an entry on EB.SUB.PRODUCT
classification	INT	See “Classifications”
systemClearFile	Y	Flag to indicate if the file should be cleared down when running the SYSTEM.CLEAR.FILES utility. Should be set to YES for any transient (i.e. financial) applications
relatedFiles	CUSTOMER CR.CAMPAIGN.DEFINITION	A space delimited list of related files. Used in creating the data model
isPostClosingFile		Y or blank. Only required when the application is required for use by Post Closing. Normally this should be left blank.
equatePrefix	CR.CD	Used to create the insert for the application that contains the equated field names
idPrefix	CRCD	If set, invokes EB.FORMAT.ID to produce transaction reference style keys

blockedFunctions		A space delimited list of functions that are not allowed on an application
triggerField		Used as the trigger field for operation processing. Refer to “Operations”.

Below is the actual code showing the above settings:

```
    Table.name = 'CR.CAMPAIGN.OPPORTUNITY'
    Table.title = 'Campaign Opportunities'
    Table.stereotype = 'H'
    Table.product = 'CR'
    Table.subProduct = 'CAMPAIGN'
    Table.classification = 'INT'
    Table.systemClearFile = 'Y'
    Table.relatedFiles = 'CUSTOMER CR.CAMPAIGN.DEFINITION'
    Table.isPostClosingFile = ''
    Table.equatePrefix = 'CR.CO'
*-----------------------------------------------------------------------------
    Table.idPrefix = 'CRCO'
    Table.blockedFunctions = ''
    Table.triggerField = ''
```

Stereotypes

There are three stereotypes to choose from depending on the type of application program:

Type	Description
Application (H or U)	Used for all standard input programs to maintain a live, unauthorised and history file. This template is also used for type ‘U’ programs that have a live and unauthorised file but do not have a history file.
Display (L)	Used for the display of a 'non-inputtable / live’ file
Utility (W)	Used to allow standard input without an unauthorised file and the verify function to execute special procedures

The stereotype of the application decides which files will be created and used. There are three files that may be created:

1. The main file is the live file or authorised file and has no suffix.
2. The unauthorised file is suffixed with $NAU and holds the record as input or changed before it has been authorised.
3. The history file is suffixed by $HIS and contains images of the record prior to each change.

The user inputs or amends data in the unauthorised file. Another user must then view the data and authorise it at which point it is moved from the unauthorised file into the live file. The existing record in the live file is moved to the history file. The infrastructure handles all the reading and writing of these files.

Classifications

There are three primary classifications of application:

Classification	Details
INT	Installation - This covers files like COMPANY, USER, LANGUAGE, where only one version of the file will exist regardless of the number of companies
CUS	Customer - for files where the data can be shared between companies. Primarily static information and tables.
FIN	Financial - for files that hold financial level details (amounts, balances etc.) where the data cannot be shared with other companies.

For further explanation and a full list of the classifications, refer to the FILE.CONTROL help text.

Non Stop Processing

Application may be NS enabled by setting the common variable C$NS.OPERATION.

Value	Meaning
ALL	Full Non Stop operation providing NS module is installed
NEW	new deals allowed in NS mode only providing NS module is installed
NOD	Nonstop input allowed with NS module (for apps like PGM.FILE, BATCH etc)
	no NS operation allowed

If the application does not allow NS there is no need to specify it as null. This allows local developments to be non-stop if required.

Further information on nonstop processing can be found in the nonstop processing user guide.

Step 2 - Define the Fields (.FIELDS)

Having defined the high level properties of the application, the next step is to define the fields that make up the application. Every application MUST have a corresponding field definitions subroutine. The name of this subroutine is the full name of the application with the suffix ".FIELDS", e.g. CR.CAMPAIGN.OPPORTUNITY.FIELDS. Your new .FIELDS subroutine should be based on the TEMPLATE.FIELDS template that is shipped as part of the release. Once defined, save and compile your .FIELDS subroutine.

Standard APIs

Table has the following methods to define the fields:

`Table.addField`	`Add a field with standard data types`
`Table.addFieldDefinition`	`Add a field using F, N and T style`
`Table.addFieldWithEbLookup`	`Add a field with a virtual table`
`Table.defineId`	`Define the ID field`
`Field.setCheckFile`	`Add a check file to a field`
`Table.setAuditPosistion`	`Defines the last field position`

Refer to the object documentation for full details. Examples of usage are below, and can also be found in the TEMPLATE.FIELDS template routine:

```
CALL Table.defineId("MESSAGE.ID", T24_String)
CALL Table.addField("TO.DAO", T24_String,"","")
CALL Table.addFieldDefinition("DISPL.APPLICATION", "1", "": FM : "Y_","")

CALL Field.setCheckFile("DEPT.ACCT.OFFICER")
CALL Table.addField("XX.MESSAGE", T24_Text,Field_Mandatory,"")
CALL Table.addField("DATE.READ", T24_Date,Field_NoInput,"")
CALL Table.addField("TO.CUSTOMER", T24_Customer, "","")

CALL Table.setAuditPosition
```

Field Settings

Each field is defined by five settings

1. Field Names and Grouping
2. Field Length
3. Field Type
4. Check file

There are also corresponding variables to define the parameters for the record id. The standard APIs hide the complexity of these items, but it is important to understand the underlying mechanism that is used in order to use the full flexibility of the field definitions.

When defining the arrays you must use the incrementing variable, Z, to reference the element of the array. In this way fields can be added later without having to renumber all the other array assignments.

The variable V is an important common variable used to hold the number of fields in the record and should be set to Z + 9 for input applications and Z for display only applications (where Z is the last field defined in the 'F' array). If the standard APIs have been used instead of direct array manipulation, this is achieved using the Table.setAuditPosition method.

Field Names and Grouping

The F array is a dimensioned array used to define the field name associated with each field and also specifies whether the field is a single value, multi-valued or sub-valued.

The corresponding variable for the record id is ID.F. The record id must be single valued.

NB the maximum size of the field name is 18 characters, which includes any multi-value definitions.

Single valued fields

Each element of the array is assigned the text string to be used on the screen to label the field. This can be up to 18 alphanumeric characters and must NOT include spaces. The first two characters cannot be 'XX' e.g.

```
ID.F = 'CHARGE.CODE'
F(Z) = 'CHARGE.TYPE'
```

Multi-valued fields

Any field (except the record id) can be multi-valued. They can be individual multi-valued fields; multi-valued in association with a language code, or part of a group of fields whose multi-values are always in association with each other. Individual multi-valued fields are defined by setting their 'F' table element to 'XX.' followed by the field name, e.g.

```
F(Z) = 'XX.NARRATIVE'
```

Language associated multi-values allow several translations of the value of the field to be held on the record, the appropriate multi-value being used according to the language code of the user. These are defined by setting the 'F' table element to 'XX.LL' followed by the field name, e.g.

```
F(Z) = 'XX.LL.DESCRIPTION'
```

Multi Value Associated Fields

Associated groups of multi-values are defined by setting the third character of the 'F' table to '<' for the first field of the association, '-' for intermediate fields and '>' for the last associated field.

```
= 'XX<CURRENCY'
= 'XX-CHARGE.RATE'
```

```
= 'XX-LOWER.LIMIT'
= 'XX>UPPER.LIMIT'
```

Sub valued fields

Any multi-valued field can be sub-valued. These can be defined individually or as associations on sub-values within each multi-value. Single valued fields or record ids cannot be sub-valued. Sub-values are defined in the same way as multi-values except characters 4 to 6 of the 'F' table elements are used instead of characters 1 to 3.

```
= 'XX<CURRENCY'
= 'XX-XX<CHARGE.RATE'
= 'XX-XX-LOWER.LIMIT'
= 'XX>XX>UPPER.LIMIT'
```

Field Length

The 'N' array is a dimensioned array that defines the maximum and minimum length of each field.

The corresponding variable for the record id is ID.N. The element is assigned with three sub-fields separated by a ".":

Sub-field 1

This parameter defines the maximum length that will be allowed on input for the field or for each multi-value or sub-value if applicable, e.g.

`N(Z)="35"` This field can have a maximum of 35 characters

If the number is prefixed by 0 (e.g. 035) then leading zeroes will not be removed. If it is prefixed by a leading space then spaces will not be trimmed; if it is not prefixed by a leading space, all leading, trailing and multiple spaces removed.

Sub-field 2

This parameter defines the minimum length that will be allowed on input for the field or for each multi-value or sub-value if applicable. Null, space or zero defines that the field is optional. If the field is mandatory, this should be set to a value of 1 (as some languages assign significant meaning to one character, and putting an arbitrary length is pointless)

Sub-field 3

Although this field is no longer used, the definition is included here for completeness. Older versions of the template defined a CHECK.FIELDS extension to do field level validation. However, with the move to a stateless architecture, this feature is irrelevant. Where this parameter is set to 'C' special editing as coded in the CHECK.FIELDS routine will be performed.

Examples:

`'10'`	Up to 10 characters allowed, no special editing will be done. Optional.
`'10.1'`	Between 1 and 10 characters allowed, no special editing will be done.
`'006.6.C'`	Input must be 6 characters. Leading zeros will not be removed and the field will be subject to processing by the CHECK.FIELDS routine.
`'35..C'`	Up to 35 characters of input will be accepted, leading spaces, trailing spaces and duplicated embedded spaces will be removed. For example, the keyed input 'LLOYDS BANK PLC' will be edited to 'LLOYDS BANK PLC.' Null will be accepted and the input will be subject to FIELD.CHECKS.

Field Type

The 'T' array is a dimensioned array that defines additional editing to be carried out when data is input in the fields.

The corresponding variable for the record id is ID.T.

Each element consists of a dynamic array. The first field identifies the name of the routine to call to perform specific editing for on data input. Fields 3, 4 and 5 are used to define display masks, input restrictions and display justification respectively. The other fields define parameters, which are used in ways specific to the particular validation subroutine and are described later in this section. The use of Fields 1, 3, 4 and 5 are described here:

Field 1 – Routine Name

Each routine is named 'IN2suffix'. This field contains 'suffix', and may be blank to invoke IN2.

Below is a summary table of the so-called “IN2 routines” that are available in T24. For more complete details refer to the IN2 Routines User Guide.

Type	Routine
Account Number	IN2ACC,IN2.ACCD,IN2ALL,IN2ANT,IN2INT
Alphabetic	IN2AAA,IN2SSS
Alphanumeric	IN2A,IN2AA,IN2S,IN2SS
Any character	IN2ANY
Amount	IN2AMT
Company Code	IN2COM
Currency Code	IN2CCY,IN2.CCYD
Customer Number	IN2CUS
Date	IN2D,IN2.ACCD,IN2.CCYD,IN2.D,IN2YM,IN2.YM,IN2FQU
Frequency	IN2FQU
Mnemonic	IN2MME
Numeric	IN2, IN2R
Program Name	IN2PG,IN2PV
Range	IN2
Rate	IN2R
Security	IN2SEC
Specific Values	IN2
Swift Characters	IN2S,IN2SS,IN2SSS
Version	IN2PV

Field 3

Defines input restrictions and can be any of the following values:

Value	Description
Null	Input will be accepted in all circumstances (subject to editing and security restrictions etc.).
NOINPUT	Input will never be allowed
NOCHANGE	Input will not be allowed once the record has been authorised
EXTERN	Input will never be allowed and the field will be cleared by the copy function

Field 4

This field is used to define a mask for input and display editing. The format of the mask is generally as used by the BASIC function FMT. A special mask is used by routines that edit dates (see the section describing these particular IN2 routines).

Field 5

Defines the display justification, Null = Left justified, R = Right justified.

Field 7

Fields to be deleted when containing default figures after input of this field when more than one field defined VM = Field marker

Field 8

Defines restrictions on multivalue fields. If used, the settings should be specified for the first field in the association and may be set to the following values:

- 'NOMODIFY' - No changes allowed to association
- 'NODELETE' - No deletion allowed to association
- 'NOEXPAND' - No expansion allowed to association

Field 9

'Hot Field' properties for browser:

- 'HOT.FIELD' – Check field validation will be performed on this field
- 'HOT.VALIDATE' – Check field validation will be performed on ALL fields (equiv. to pressing browser 'Validate' button)

Virtual Tables

The concept of virtual tables has been added to prevent the growth in the number of simple tables that are shipped as part of T24 which serve only to provide a list of options that Temenos cannot hard code, e.g. statuses, titles, etc.

Rather than create a new table for each of these, a virtual table can be defined.

```
VIRTUAL.TABLE.LIST = ''
VIRTUAL.TABLE.LIST = 'VIRTUAL.TABLE'
CALL EB.LOOKUP.LIST(VIRTUAL.TABLE.LIST)
```

The variable VIRTUAL.TABLE.LIST can then be used to directly populate the T array directly:

```
T(Z) = VIRTUAL.TABLE.LIST                    ; * List from EB.LOOKUP
```

The actual items for the virtual table are defined on one table, called EB.LOOKUP. The key to this table is VIRTUAL.TABLE.NAME*REAL.KEY, e.g. EB.STATUS*COMPLETE.

The list populates the second field of the T array, such that a list of the available options is available to the user without need to use a dropdown. EB.LOOKUP contains additional fields in a name – value pair structure that may be used to contain parameter information. Refer to the help text on EB.LOOKUP for more information.

Operations

Provision has been added to allow OPERATION style fields to be defined. The concept is that a trigger field is used to determine the action that is being performed, and depending upon that action certain fields in the application will be made available for input while others will become no input. This concept has existed in T24 applications for some time (e.g. in the LC and PD modules) and now support has been added to the template.

Firstly, a field must be defined that holds the options, and should be named OPERATION

```
Z+=1 ; F(Z) = "OPERATION" ; N(Z) = "35" ; T(Z) = "" ; T(Z)<2> = 'ACCEPT_REJECT'
```

The property `Table.Trigger` is used to store which field will act as the trigger for the operation process, and should be set to the equated field name:

```
Table.Trigger = MY.APP.OPERATION
```

There are two further properties that are used to control which fields will be made available for input, and which fields will be set to no input. Each property takes a value mark (`VM`) delimited list of fields.

```
ACCEPT.FIELDS.NOINPUT = MY.APP.REJECT.DATE : VM : MY.APP.REJECT.REASON
REJECT.FIELDS.INPUT = MY.APP.REJECT.DATE : VM : MY.APP.REJECT.REASON
accept = 1
reject = 2
Table.inputtableFields<reject> = REJECT.FIELDS.INPUT
Table.noInputFields<accept> = ACCEPT.FIELDS.NOINPUT
```

In the above example, the fields REJECT.DATE and REJECT.REASON will be no input when the ACCEPT operation is selected – all other fields will behave as defined in the T array. When the REJECT operation is selected, only the REJECT.DATE and REJECT.REASON fields will be available for input – all others will no input. If both properties are set for an operation, the C_NOINPUTS property is ignored. NB If we are processing a NOINPUTS list, then any field that is already defined as NOINPUT, NOCHANGE, etc. will retain that definition to prevent reserved fields, etc. being made inputtable! The variables accept and reject refer to positions in the list of allowed operations (`T(Z)<2>`).

CHECKFILE Array

This array is a dimensioned array that defines file look-ups to be performed when data is input. The corresponding variable for the id is ID.CHECKFILE. For new applications, the API `Field.setCheckFile` should be used, and the information here is for reference to older applications built before the new APIs were available. As such, DO NOT use the CHECKFILE array for new applications.

The check file processing will validate the input as having an existing record on the file and will also retrieve the specified field of that record which is then displayed as enrichment next to the entered field. The elements of the check file table are a dynamic array that is set up as follows:

Field 1

The application name of the file to be checked, e.g. CUSTOMER.

Field 2

The number of the field to be returned as enrichment or to be used as the id for the next file read. This MUST be a file name defined in the corresponding file layout for the check file.

Field 3

This field is used to define 4 optional parameters. These are separated by '.' and should be assigned as follows:

Subfields 1, 2 and 3

Null. These subfields are not used.

Subfield 4

'D' The id is date associated and the record with the most recent date should be used. The date is in YYYYMMDD format.

'YM' As for 'D' but the date is in YYYYMM format.

null =No date used.

Example

To encourage re-use, check file definitions for tables should be defined in the initialise section of the .FIELDS and then referenced later.

```
CHK.ACCOUNT = "ACCOUNT": FM : AC.SHORT.TITLE: FM :"L"
CHECKFILE(Z) = CHK.ACCOUNT
```

Standard Fields

All new applications MUST define a set of reserved fields that can be used to add extra fields to the application without the need to change the layout of the data. You should add ten reserved fields, the names of which should be RESERVED.1 etc. The standard API for this is:

```
Table.addReservedField(fieldname)
```

Below are the F, N and T array setting for use in older templates:

```
Z+=1 ; F(Z) = "RESERVED.10" ; N(Z) = "35" ; T(Z) = "" ; T(Z)<3> = 'NOINPUT'
```

All applications MUST contain a LOCAL.REF field. This allows T24 clients to add user definable fields to the application. The standard API for this is:

```
Table.addLocalReferenceField
```

Below are the F, N and T array setting for use in older templates:

```
Z+=1 ; F(Z) = "XX.LOCAL.REF" ; N(Z) = "35" ; T(Z) = "A"
```

All applications MUST have a field to store overrides. Even if your application does not currently use the override processing (refer to the “Adding Business Logic” section), it is probable that it will in the future. The standard API for this is:

```
Table.addOverrideField
```

Below are the F, N and T array setting for use in older templates:

```
Z+=1 ; F(Z) = "XX.OVERRIDE" ; N(Z) = "35" ; T(Z) = "" ; T(Z)<3> = 'NOINPUT'
```

Applications that raise accounting entries MUST define a field to hold the entry ids that have been raised. This MUST be the last field that you define. The standard API for this is:

```
Table.addStatementNumbersField
```

Below are the F, N and T array setting for use in older templates:

```
Z+=1 ; F(Z) = "XX.STMT.NOS" ; N(Z) = "35" ; T(Z) = "" ; T(Z)<3> = 'NOINPUT'
```

Applications that raise delivery events must define a field to hold the delivery references. By default, the name of this field should be DELIVERY.REF, though you may need to have multiple fields to hold delivery references for multiple involved parties. The standard API for this is:

```
Table.addDeliveryReferenceField
```

Below are the F, N and T array setting for use in older templates:

```
Z+=1 ; F(Z) = "XX.DELIVERY.REF" ; N(Z) = "35" ; T(Z) = "" ; T(Z)<3> = 'NOINPUT'
```

Refer to the Table API documentation for further details on the APIs to add these standard fields.

Example Fields Types – APIs

Below if an example list of field types using the Table API

```
* Definition of the key
CALL Table.defineId("TABLE.ID", T24_String)

*A standard numeric field
CALL Table.addField("NUMERIC", T24_Numeric, "", "")

*Alpha numeric. See also T24_Big_String
CALL Table.addField("ALPHA", T24_String, "", "")

* A text box
CALL Table.addField("TEXT", T24_Text, "", "")

* Limited options
CALL Table.addYesNoField("YES.OR.NO", "", "")
CALL Table.addOptionsField("LIST", "OPTION1_OPTION2_OPTION3", "", "")

* A virtual table look based on EB.LOOKUP
CALL Table.addVirtualTableField("MOOD", "CR.MOOD", "", "")

* CUSTOMER field
CALL Table.addField("CUSTOMER.NO", T24_Customer, "", "")

* Account field
CALL Table.addField("ACCOUNT.NO", T24_Account, "", "")

* Only internal accounts
CALL Table.addField("INTERNAL.NO", T24_InternalAccount, "", "")

* Currency field
CALL Table.addField("CURRENCY", T24_String, "", "")
CALL Field.setCheckFile('CURRENCY')

* Allows all amounts
CALL Table.addAmountField('DEBIT.AMT', 'CURRENCY', Field_AllowNegative, '')

* Only positive amounts
CALL Table.addAmountField('DEBIT.AMT', 'CURRENCY', '', '')

* Category field
CALL Table.addField("CATEGORY", T24_Numeric, "", "")
CALL Field.setCheckFile('CATEGORY')

* Delivery Ref field
CALL Table.addDeliveryReferenceField

*-----------------------------------------------------------------------------
* Example block of reserved fields
*-----------------------------------------------------------------------------

CALL Table.addReservedField('RESERVED.10')
CALL Table.addReservedField('RESERVED.1')
*-----------------------------------------------------------------------------
CALL Table.addLocalReferenceField
CALL Table.addOverrideField
*-----------------------------------------------------------------------------
CALL Table.addStatementNumbersField
*-----------------------------------------------------------------------------
CALL Table.setAuditPosition
```

Example Fields Types - Arrays

Below if an example list of field types showing the F, N and T arrays.

```
* Definition of the key
ID.F = "BLANK.ID" ; ID.N = "35" ; ID.T = "A"

Z=0
*A standard numeric field
Z+=1 ; F(Z) = "NUMERIC" ; N(Z) = "35" ; T(Z) = ""

*Alpha numeric. See also AA, AAA, S, SS & SSS
Z+=1 ; F(Z) = "ALPHA" ; N(Z) = "35" ; T(Z) = "A"

* A text box
Z+=1 ; F(Z) = "XX.TEXT" ; N(Z) = "35" ; T(Z) = "A" ; T(Z)<7> = 'TEXT'

* Limited options
Z+=1 ; F(Z) = "LIST" ; N(Z) = "35" ; T(Z) = "" ; T(Z)<2> = 'OPTION1_OPTION2_OPTION3'

* List from EB.LOOKUP
Z+=1 ; F(Z) = "VIRTUAL.TABLE" ; N(Z) = "35" ; T(Z) = VIRTUAL.TABLE.LIST

* CUSTOMER field
Z+=1 ; F(Z) = "CUSTOMER" ; N(Z) = "35" ; T(Z) = "CUS" ; CHECKFILE(Z) = CHK.CUSTOMER

* Account field
Z+=1 ; F(Z) = "ACCOUNT" ; N(Z) = "35" ; T(Z) = "ACC" ;  CHECKFILE(Z) = CHK.ACCOUNT

* Only internal accounts
Z+=1 ; F(Z) = "INTERNAL.ACCOUNT" ; N(Z) = "35" ; T(Z) = "ANT" ; CHECKFILE(Z) = CHK.ACCOUNT

* Currency field
Z+=1 ; F(Z) = "CURRENCY" ; N(Z) = "35" ; T(Z) = "CCY" ; CHECKFILE(Z) = CHK.CURRENCY

* Allows all amounts
Z+=1 ; F(Z) = "AMOUNT" ; N(Z) = "35" ; T(Z) = "AMT" ; T(Z)<2,1> = '-'; T(Z)<2,2> =
MY.APP.CURRENCY

; * Only positive amounts
Z+=1 ; F(Z) = "POSITIVE.AMOUNT" ; N(Z) = "35" ; T(Z) = "AMT" ; T(Z)<2,1> = ''; T(Z)<2,2> =
MY.APP.CURRENCY

* Category field
Z+=1 ; F(Z) = "CATEGORY" ; N(Z) = "35" ; T(Z) = "" ; CHECKFILE(Z) = CHK.CATEGORY
* Delivery Ref field
Z+=1 ; F(Z) = "XX.DELIVERY.REF" ; N(Z) = "35" ; T(Z) = "" ; T(Z)<3> = 'NOINPUT'

*-----------------------------------------------------------------------------
* Example block of reserved fields
*-----------------------------------------------------------------------------

Z+=1 ; F(Z) = "RESERVED.10" ; N(Z) = "35" ; T(Z) = "" ; T(Z)<3> = 'NOINPUT'
Z+=1 ; F(Z) = "RESERVED.1" ; N(Z) = "35" ; T(Z) = "" ; T(Z)<3> = 'NOINPUT'
*-----------------------------------------------------------------------------
Z+=1 ; F(Z) = "XX.LOCAL.REF" ; N(Z) = "35" ; T(Z) = "A"
Z+=1 ; F(Z) = "XX.OVERRIDE" ; N(Z) = "35" ; T(Z) = "" ; T(Z)<3> = 'NOINPUT'

*-----------------------------------------------------------------------------
Z+=1 ; F(Z) = "XX.STMT.NOS" ; N(Z) = "35" ; T(Z) = ""
*-----------------------------------------------------------------------------
V = Z + 9
```

Step 3 – Artefact Creation using EB.DEV.HELPER

EB.DEV.HELPER is a standard utility that creates all of the artefacts that are required to run an application in T24. The key to the record is application that you are working on, and once verified will perform a number of actions if relevant field is flagged

PGM.FILE	Creates a PGM.FILE record based on the information set in the properties array
FILE.CONTROL	Creates a FILE.CONTROL record based on the information set in the properties array
INSERT	Creates the insert file (I_F.MY.APPLICATION) that holds the equated field names. Uses the prefix defined in the EQUATE.PREFIX property
CREATE.FILE	Creates the physical database files using EBS.CREATE.FILES
UPDATE.SS	Updates / Creates the STANDARD.SELECTION record via OFS and sets the REBUILD.SYS.FIELDS flag to 'Y'
CREATE.DAS.PGMS	Creates the DAS items in BP for the application: DAS.application I_DAS.application I_DAS.application.NOTES

If `Table.equatePrefix` is defined, then an EB.SYSTEM.ID record will be created if one does not already exist.

NB Once created these artefacts are still required to be shipped with your code. EB.DEV.HELPER merely aids the developer in creating the items.

Step 4 – Adding Business Logic to the Application

T24 allows extensions to be added to customise the application. This section details the extension points that are available, and their purpose. Refer to the appendix “Program Flow” for details of how and when the extension points are invoked.

Each extension point has a corresponding template that is shipped as part of the T24 release, and prefixed with TEMPLATE, e.g. TEMPLATE.ID, TEMPLATE.RECORD, etc. and code examples and TODO hints are included in these templates.

Refer to “Programming Standards User Guide” for further details on coding conventions and standards.

In order for the extension point to be invoked, the relevant routine only has to exist.

Extension	Routine suffix
Check id	ID
Check Record	RECORD
Cross validation	VALIDATE
Field definitions	FIELDS
Overrides	OVERRIDES
Delivery Preview	PREVIEW
Main processing	PROCESS
Work that needs to be done at authorisation	AUTHORISE
Check Function	FUNCTION
Work routine (type W only)	RUN
Field Defaulting	DEFAULT
Initialisation	INITIALISE

- You MUST write a .FIELDS routine for the field definitions
- For type W applications, you MUST write a .RUN routine

Initialisation (.INITIALISE)

This section should be used to perform any initialisation that needs to be done once only when the application is first entered. Typically, this includes opening files and reading parameter records into common.

Checking the Function (.FUNCTION)

It may be necessary to prohibit certain functions for the particular application or to perform special checks if such functions are attempted. The verify function is blocked, unless the stereotype of the application is utility (W). Unconditional function blocks can be set in the `Table.blockedFunctions` property.

Applications should support all functions, but there may be circumstances that prevent certain functions from being carried out.

If you want to block a function, the error message should be set to "**EB-FUNT.NOT.ALLOWED.APP'**" which is the key to an error in the EB.ERROR application.

Checking the ID (.ID)

This routine is invoked immediately after a record id has been entered and any special editing of the record id should be coded here.

Transaction reference style ids can be validated using EB.FORMAT.ID by simply setting the `Table.idPrefix` property.

The value entered will be in ID.NEW. If an error is found requiring the id to be re-input, assign an error message assigned to E and again the error should be the key to an EB.ERROR record. For rules and examples of how define the key to EB.ERROR is defined in *T24 Programming Standards*, e.g.

```
    APPL.ID = FIELD(ID.NEW,',',1)
    CALL IN2PV(ID.N,"PV":FM:"HULWD")  ; * should be a valid version or application
*
* if its not a valid version or application then it should be 'SYSTEM'
*
    IF ETEXT <> "" AND ID.NEW <> 'SYSTEM' THEN ;
        E = ETEXT ; * if either of these, then error
        ETEXT = ''
    END
```

Note that at this point the record has not been read. Any validation that requires the record

Checking the Record (.RECORD)

The record has been read from file when this section is performed and any special editing of the record can be done here. The record will be assigned to one or more fields of R.NEW. If a condition is found requiring the id to be re-input a message should be assigned to E.

Also allows you to block a function given the state of a record, e.g. don't allow input once a payment has been made. Again, control this by setting E.

It is possible to modify the N and T arrays of the application in this section (for instance to make certain fields no-input based on certain criteria) however this is not recommended. Instead, consider using NOCHANGE fields and / or the trigger field functionality.

Validation (.VALIDATE)

This section will be performed only when function key F5 has been entered and V$FUNCTION is 'I' or 'C'.

This should contain all the code required to fully validate the record being input or changed. R.NEW will contain the record as input or amended during the current transaction, R.NEW.LAST will hold the record as it was before any input or changes were made and R.OLD will contain the record as it currently stands on the live file.

Fields in error can be flagged with a corresponding error message by calling STORE.END.ERROR with AF, AV and AS set to define the offending field and ETEXT to the message, a key to the EB.ERROR application. The section would normally process all the fields of the record so that the user can see all errors in one attempt.

However, some cross-validation will make it futile checking further fields if related fields are in error. The decision on how to handle the cross-validation must depend on the particular application.

Be aware that this routine will be invoked for DELETE and REVERSE as well. Check V$FUNCTION to determine what is being invoked.

You must NEVER update files in this routine.

Override Processing (.OVERRIDES)

Fields that require overrides (do not embed this in the .VALIDATE routine) should be analysed in this subroutine. Any override messages should be displayed by calling STORE.OVERRIDE(CURR.NO) and setting TEXT to the message (do not modify the value of CURR.NO); text should be set as a key to a record in the OVERRIDE application. If TEXT is returned as NO, then exit the subroutine immediately. The subroutine should initialise the override fields by calling STORE.OVERRIDE with CURR.NO set to zero at the start of the subroutine.

Processing (.PROCESS)

This section should contain coding to perform processing which is to be done before the record is written to the unauthorised file. It will be executed when V$FUNCTION is D, R, C or I and the .VALIDATE routine has performed successfully. Typical use of this section would be to make related updates to other files, call accounting, etc.

Authorisation (.AUTHORISE)

This section is used for any special processing required before the live record is written. It is normally used to invoke accounting in authorisation mode and to handle the authorisation of a reversal.

Where there are multiple authorisations, this routine will be called multiple times. This routine is also invoked during the authorisation of a reversal. To determine the real action, interrogate the RECORD.STATUS.

Step 5 – Creating the Data Access Service

Introduction

All queries (i.e. SELECT operations) that are performed against a table are consolidated into a Data Access Service (DAS) for the application. This centralises all query processing for a given table in one place and removes any query language from the code. Each query is expressed in a simple meta-query language (as per ENQUIRY) that defines the fields, operands, data and joins between criteria (AND / OR).

When executed, the meta-query language generates the specific query language depending on the target database (JQL, SQL, xQuery, etc.)

This approach allows control on the DAS routines and prevents poor queries being defined. Each query in the DAS is named and has a description, and the DAS itself allows discovery of the queries that are supported. There is also a facility to cache the results of static data.

The name of the DAS is DAS.MYAPPLICATION, where MYAPPLICATION is the name of the application, e.g. DAS.EB.LOOKUP

Defining the Queries

Each query definition has two parts – the name of the query and a description of the query.

The name of the query is a simple numeric equate and is stored in I_DAS.MYAPPLICATION:

```
EQU dasExamplesInterestSchedules TO 1
EQU dasExamplesTodaysSchedules TO 2
EQU dasExamplesDealsOfAType TO 3
EQU dasExamplesCustomerCurrency TO 4
```

Each query name must be equated to a unique integer, as this is the pointer used in the caching mechanism. The name of the query must be prefixed with DAS.MYAPPLICATION$, and then followed by a meaningful name.

The description of the query is held in I_DAS.MYAPPLICATION.NOTES, which should be created from the template I_DAS.TEMPLATE.NOTES as it must contain a dimensioned array for the cache and the notes.

```
COMMON/DAS.EXAMPLES/DAS$CACHE(100),DAS$NOTES(100)
*-----------------------------------------------------------------------------
DAS$NOTES(dasExamplesInterestSchedules) = 'All interest Schedules'
DAS$NOTES(dasExamplesTodaysSchedules) = 'All schedules with today's date'
DAS$NOTES(dasExamplesDealsOfAType) = 'All deals with the supplied DEAL.TYPE'
DAS$NOTES(dasExamplesCustomerCurrency) = 'Deal that have the supplied customer AND
supplied currency'
```

Implementing the Queries

The implementation of each query is in the DAS subroutine itself. This must be created from the template DAS.TEMPLATE, and each query is an additional CASE statement based on MY.CMD, which holds the name of the query.

In addition to the queries defined specifically for the application, the generic query `DAS$ALL.IDS` is defined to return all the keys on the table. It is only necessary to include this query in the DAS if this list can be cached, i.e. the contents of the table are static at run time.

```
CASE MY.CMD = DAS$ALL.IDS ; * Standard to return all keys
   ADD.TO.CACHE = 1
```

Simple queries define the fields, operands and the data to query. The data may be literals:

```
CASE MY.CMD = dasExamplesInterestSchedules;* All interest schedules
   MY.FIELDS = 'SCHEDULE.TYPE'
   MY.OPERANDS = 'EQ'
   MY.DATA = 'INTEREST'
```

Common variables:

```
CASE MY.CMD = dasExamplesTodaysSchedules;* All schedules due today
   MY.FIELDS = 'SCHEDULE.DATE'
   MY.OPERANDS = 'EQ'
   MY.DATA = TODAY
```

Or use variable criteria that is supplied when the DAS is invoked:

```
CASE MY.CMD = dasExamplesDealsOfAType ;* Contracts of a certain DEAL.TYPE
   MY.FIELDS = 'DEAL.TYPE'
   MY.OPERANDS = 'EQ'
   MY.DATA = THE.ARGS
```

More complex queries are defined by adding fields to the arrays, and specifying the MY.JOINS array:

```
CASE MY.CMD = dasExamplesCustomerCurrency; * For a customer with given CCY
   MY.FIELDS = 'CUSTOMER'
   MY.OPERANDS = 'EQ'
   MY.DATA = THE.ARGS<1>
   MY.JOINS = 'AND'
   MY.FIELDS<2> = 'CURRENCY'
   MY.OPERANDS<2> = 'EQ'
   MY.DATA<2> = THE.ARGS<2>
```

Caching Query Results

The DAS infrastructure allows the caching of results by setting:

```
ADD.TO.CACHE = 1
```

Once the query results have been returned, they are added to the cache mechanism and further invocation of the named select will always use the cached results. The cache must ONLY be used where the result of the query is static at run time, e.g. CATEGORY or COMPANY. Caching must NOT be used where variable data is used in the selection.

Testing the DAS

By invoking the DAS in discover mode, every query that is defined is exercised and dummy query statements are created, though not exercised:

```
jsh r06  -->TEST.DAS EXAMPLE
All interest schedules.
SELECT F.EXAMPLE WITH SCHEDULE.TYPE EQ INTEREST

All schedules with today's date.
SELECT F.EXAMPLE WITH SCHEDULE.DATE EQ 20060426

All deals with the supplied DEAL.TYPE
SELECT F.EXAMPLE WITH DEAL.TYPE EQ ARG1

Deals that have the supplied customer AND the supplied currency
SELECT F.EXAMPLE WITH CUSTOMER EQ ARG1 AND CURRENCY EQ ARG2

jsh r06  -->
```

Invoking the DAS

The public API for each Data Access Service is the subroutine DAS. For full details, refer to the documentation of that routine. However, here is an example of a DAS being invoked:

```
...

$INSERT I_DAS.EB.LOOKUP
...
    THE.ARGS = VIRTUAL.TABLE
    THE.LIST = DAS.EB.LOOKUP$ITEMS
    CALL DAS('EB.LOOKUP', THE.LIST, THE.ARGS,'')
```

Other Considerations

EURO Conversion Issues

New applications with accounts, local currency amount fields or exchange rate fields need to be defined in the EURO module. Refer to the Euro manual for details.

Multi Company Processing

The infrastructure incorporates the concept of multi-company processing whereby branches (or any 'legal vehicle') operating on the system can share certain data files (customer files, tables etc.) whilst still controlling their own financial files. The rules to determine which files can be shared are defined by the application programmer when creating the application, the user can determine the company infrastructure i.e. who shares what.

All files accessed by the system are opened by a standard procedure which, using the rules and existing company infrastructure, determines which disk file should be opened. Hence multiple companies can be set up without modifications to the application code.

In an extended multi company environment branches or (any “legal vehicle”) will be linked to a Lead company (the direct equivalent of a multi company) and will share all data files with the Lead company, with the exception of certain data files used for producing financial reports (file classification FRP). It is important when accessing parameter files that the core routine EB.READ.PARAMETER is used to ensure that the correct record is read. For example reading parameter files with a key of ID.COMPANY should not be used as in extended multi company the record will not be present for a branch. Also it is bad practice to attempt to open a file by setting up the mnemonic within the code, except in very special circumstances and in this case the mnemonic should be that of a lead company. NB: on a company record the fields FINANCIAL.COM and FINANCIAL.MNE will indicate the lead company of a branch, and thus the company that owns the financial level (FIN) files.

The Infrastructure

Introduction

The T24 infrastructure provides many features so that the developer does not need to write or even consider these items. This section gives an overview of these features.

Screen Management

One of the most important aspects of the infrastructure is its screen management capability. Once the application has defined the input fields for the transaction, the infrastructure will present a standard screen for input, display, authorisation, history comparison etc. It will also handle field positioning, page manipulation, data formatting, input enrichment etc. which means that whilst the applications may differ dramatically, the user will always be presented with a standard mechanism for entry or data manipulation. However, this does not mean that the user is constrained to one design of entry screen. The infrastructure also allows a 'version' of the screen to be defined without any coding changes necessary at the application end (see the *Version System Administration Guide*). The data is presented both in ‘classic’ text based screens and in T24 Browser: neither type of display requires modifications to the application program.

Data Entry Functions

The infrastructure provides all the necessary transaction processing functions necessary to complete an application.

- Input a record
- Modify an existing record
- Display a record
- Delete a record
- Copy a record
- Authorise a record
- Reverse an authorised record
- Compare history records
- Restore a record from history
- List all or a selection of records

In the simplest case, to achieve this functionality the programmer would simply have to enter the field definitions. For more complicated processing the extension points for the specialised code necessary for authorisation, deletion are needed.

Security Management

The infrastructure provides two important aspects of security management, Access restriction and Activity logging.

System access can be controlled on four levels: system sign-on (including time restrictions), entry to applications and data records, restricting functionality (e.g. allowing display only) and defining the field values with can be entered (e.g. amount < 10,000).

Logging can be specified from simply recording sign on/off times to recording every access the user makes.

The application program requires no special code to achieve this functionality. See the *Security Management System Administration Guide* for more details.

Validation

Input validation can be specified with the field definitions and hence the infrastructure will check the data entered. The validation available is comprehensive, offering simple checks such as numeric/non-numeric input, to complicated date and amount edit checks. Input can also be verified against an existing table as well as being passed to an application specific routine.

To allow maximum flexibility, the infrastructure can pass control back to the application for further validation at any stage. The template also contains sections for further validation on completion of input (cross-validation), authorisation, deletion etc.

Main File Maintenance

The infrastructure in addition to controlling the transaction input also controls the update of the main transaction file. In this way it can maintain both authorised and unauthorised versions of a record and keep a log of the last user to input/change the data (this it stores in the data record itself).

History Maintenance

As an extension of the main file update procedure, the infrastructure will optionally maintain a history of all changes made to the application records. The user will then have the ability to 'walk through' the history file examining every change as it occurred and even restore the last record from history in the case of accidental reversal.

Transaction Journaling and System Recovery

Recovery and transaction management is provided by the T24 infrastructure. Special coding is not required.

The recovery system is based on jBASE transaction management. The physical updates to the database files do not take place until the end of the transaction, i.e. after the commit transaction is performed.

It is possible to use the JOURNAL file to store information pertaining to all the writes that occur during transactions input. By default this information is not captured and has to be setup by using the SPF application and setting the field INFO.JOURNAL to Y.

To roll back logically within a transaction, due to a program bug or operator override, the procedure has to abort the current transaction and return to the start of the transaction (id input or whatever). This is possible because no updates have actually taken place.

Another advantage of this mechanism is any application errors that cause a program to abort and would normally involve a system restore/roll forward, can be ignored in terms of data integrity.

Refer to “Backup, Restore and Recovery System Administration” guide.

Audit Trail

All main file updates are 'stamped' by the infrastructure with the name of the inputter and authoriser, date-time, terminal number, company code etc. to provide a comprehensive audit trail. If history is maintained then the audit trail will be carried in the history records as well, providing a log of every update performed.

Close of Business Processing

A batch system is provided to control all Close of Business processing. It provides the operator with a controlled environment which handles job scheduling/frequency/ dependency, report routing etc.

Applications can define Close of Business processes (routines and/or operating system commands), which are incorporated into a 'command stream', executed by the operator. The environments (company processing details etc.) are initialised by the batch control system so that application programs can be written independently of site configuration.

Appendices

Appendix 1 - Glossary

Term	Description
Domain	Defines areas of business functionality, e.g. Retail, Treasury, etc.
Product	A specific piece of business functionality within a domain, e.g. Funds Transfer, Foreign Exchange. Domains are made of multiple Products
Module	Same a product
Application	A program or subroutine that allows data entry into a T24 file, e.g. CUSTOMER, FOREX etc. Products will consist of many applications.
Subroutine	A routine invoked by an application program or close of business process, i.e. not directly executable by the user.
Infrastructure	The main supporting sub-system for an application. It manages screen input, security (SMS), file updates etc.
Live file	The file which holds authorised data.
Unauthorised file	The file which holds unauthorised data.
History file	The file which contains copies of previously authorised data.
Concat file	A file which is used as an alternate index to a live file. These should not be used unless there is a specific reason why an index would not be suitable.

Appendix 2 - Development Artefacts

FILE.CONTROL

FILE.CONTROL acts as the registry for tables for T24. Refer to helptext for further information.

PGM.FILE

PGM.FILE acts as the registry for applications and programs in T24. Refer to the helptext for more information.

STANDARD.SELECTION

STANDARD.SELECTION is the dictionary application for T24 and each application must have an entry in STANDARD.SELECTION so that the T24 framework will function correctly.

The STANDARD.SELECTION record can be built from the underlying code by setting the field REBUILD.SYS.FIELDS to Y and committing the record.

EB.SYSTEM.ID

EB.SYSTEM.ID is used to provide a description of the SYSTEM.ID field in the accounting entry files, STMT.ENTRY, CATEG.ENTRY, CONSOL.NET.TODAY and RE.CONSOL.SPEC.ENTRY.

The application also holds additional details of the underlying application record which was responsible for raising the entries and is used to allow the correct drill-down in entry based enquiries.

Insert File (I_F.XXX)

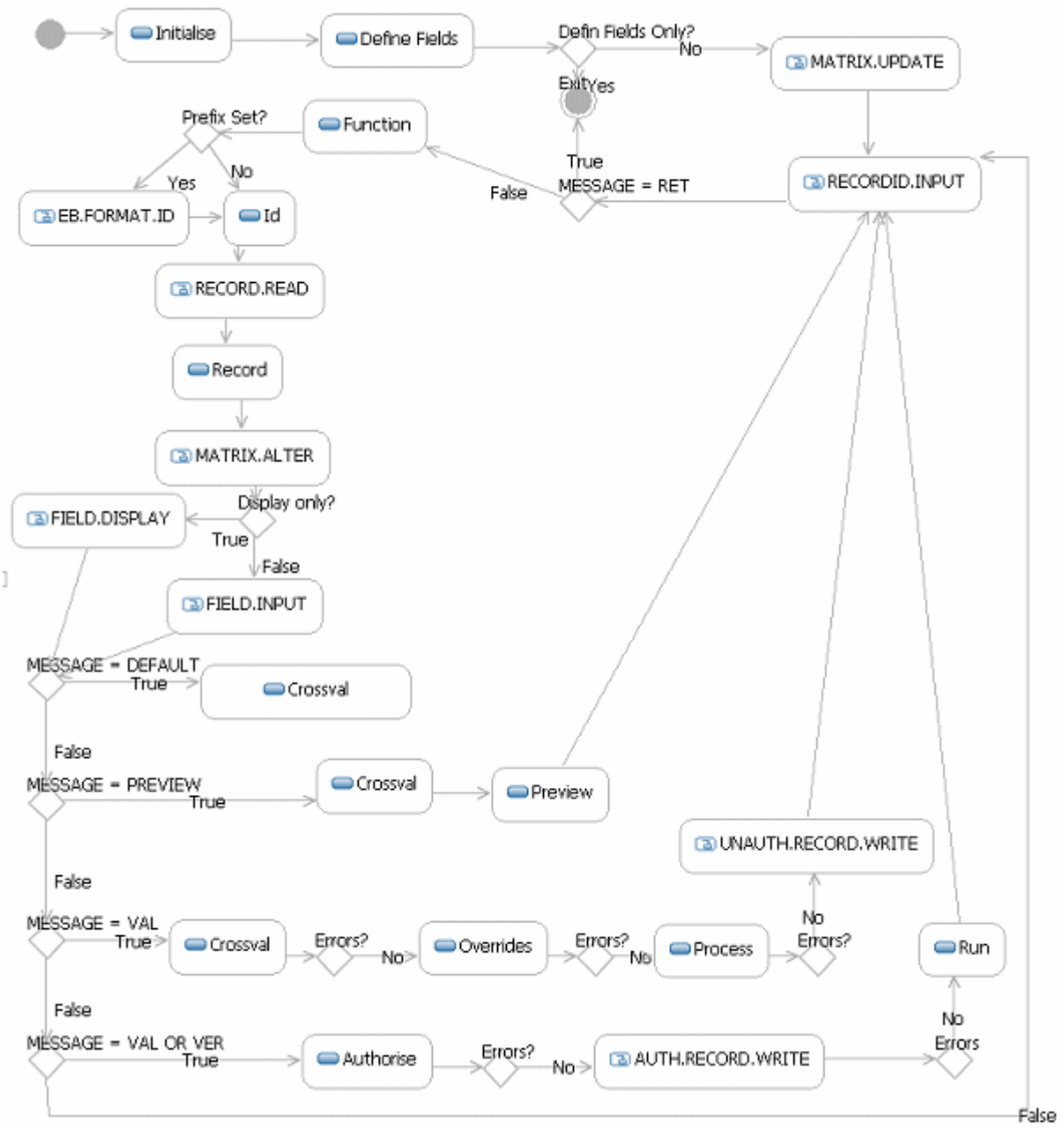
The “insert file” is the termed used to describe the generated file that holds the equated field names. This allows field names to be used in the code to reference fields without the need to change these when the table layout changes. It is created automatically from EB.DEV.HELPER, though for older applications this must be done using the command line facility FILE.LAYOUT.

DAS programs

Acts as an entry point for all defined queries on the table. Refer to T24 Application Development and Subroutine User Guides.

Appendix 3 - Program Flow

The flow of logic through the template code.



- Items with a solid blue lozenge are available for programmers to add there own code.
- Items with a locked symbol cannot/must not be changed.

Define Fields Only?

If the length V$FUNCTION greater than 1, then the application has been invoked to only populate the field definitions. Several system utility programs call application subroutines solely to assign the parameter tables for their own use. These calls will exit here.

MATRIX.UPDATE

Sets up the internal tables and variables from the parameters assigned in the .FIELDS routine

RECORDID.INPUT

This routine handles the input of the FUNCTION to be performed (input, authorise, list etc.) and the input of the record key. It sets MESSAGE to 'RET' if the application is to be exited or to 'NEW FUNCTION' if the FUNCTION has been initially input or changed.

MESSAGE = RET?

If MESSAGE is set to “RET” then the application exits.

Prefix Set?

If a prefix is set in the properties, EB.FORMAT.ID is invoked to validate the ID entered against the prefix. Used for standard transaction reference style ids.

RECORD.READ

This subroutine reads the records from the files into one or more of R.NEW, R.NEW.LAST and R.OLD, depending on the function being used.

Where the function is I or A, if an unauthorised record exists it will be assigned to R.NEW and R.NEW.LAST, otherwise the live record will be assigned to R.NEW and R.NEW.LAST, unless it is also not present, in which case the arrays will be initialised to nulls. R.OLD will be assigned with the live record if it exists, otherwise with nulls.

Where the function is ‘C’, if an unauthorised record exists it will be assigned to R.NEW otherwise R.NEW will be assigned from the live record. R.NEW.LAST and R.OLD will be set to nulls.

If the function requires a record to be present and it is not found MESSAGE will be set to 'REPEAT'.

MATRIX.ALTER

This subroutine does further initialising of the internal parameters dependant on the size and infrastructure of the record, which has been read.

Display only?

If the application is display only (`FILE.TYPE EQ 'I'`) the input routines are used, if not the display routines are used. If the application is for a live only file (PGM.TYPE 'L') the program will not allow input.

FIELD.DISPLAY / FIELD.MULTI.DISPLAY

FIELD.DISPLAY is used to display fields when multiple fields per line are not specified, whereas FIELD.MULTI.INPUT is used to display fields when multiple fields per lines are specified.

FIELD.INPUT / FIELD.MULTI.INPUT

FIELD.INPUT processes all field input when no multiple fields per line are defined by VERSION, whereas FIELD.MULTI.INPUT is used when the screen has been defined as multiple fields per line by means of a VERSION.

The MESSAGE Variable

This section of code is entered when the user has ended the field input session by entering the function key F5. MESSAGE is set to 'VAL' if the record was being input or changed and required validation; to 'AUT' if the authorise function is being used.

UNAUTH.RECORD.WRITE

This system routine writes the input or changed record to the unauthorised ($NAU) file.

AUTH.RECORD.WRITE

This system routine writes the input or changed record to the live file.

Appendix 4 – Older Templates

Introduction

T24 applications have seen three previous templates. This section gives a brief history of these and explains some of the older features that will be seen in applications based on these previous templates. For a full understanding of how these different releases of the template work, refer to the user guides “Template Maintenance” and “T24 Application Development R08”.

Template V 1

The original template identified different sections and broke these out into labels, though these were accessed in a drop down manner. Most code for an application was in a single subroutine. Each new application is a copy of a template.

Template V 2

The structure changes such that subroutines are used and the flow is controlled using GOSUB statements rather than GOTOs, leading to a much easier to understand template. Each new application copies the template code, and the template code itself may be modified. Most code for an application is still in a single subroutine which leads to some very large and complex programs. Conflicts happen when multiple people need to work on a single piece of code.

Template V 3

Very similar to the second version, but here each extension point was written as a separate routine and templates were issued against these. The XX.CROSSVAL template automatically repeated check fields at the cross validation step.