Debugging

It happens that even the best developer makes errors. It happens that the best developer of some reason do not get the code right first time. It actually happens often for most developers.

Thus the typical developer, develop according to what is defined as a waterfall model:

- Analyze
- Design
- Code
- Test
- Evaluate
- Document

And then these steps are iterated (as a whole or some of them) until the application contains the features it should have and the bugs have been ironed out.

This whitepaper will describe the different techniques used to make debugging as painless as possible.

The techniques described in this document can often be used for other situations than those listed, but if the documentation is followed, you will have a good idea about where the problem originates and what to do to solve it.

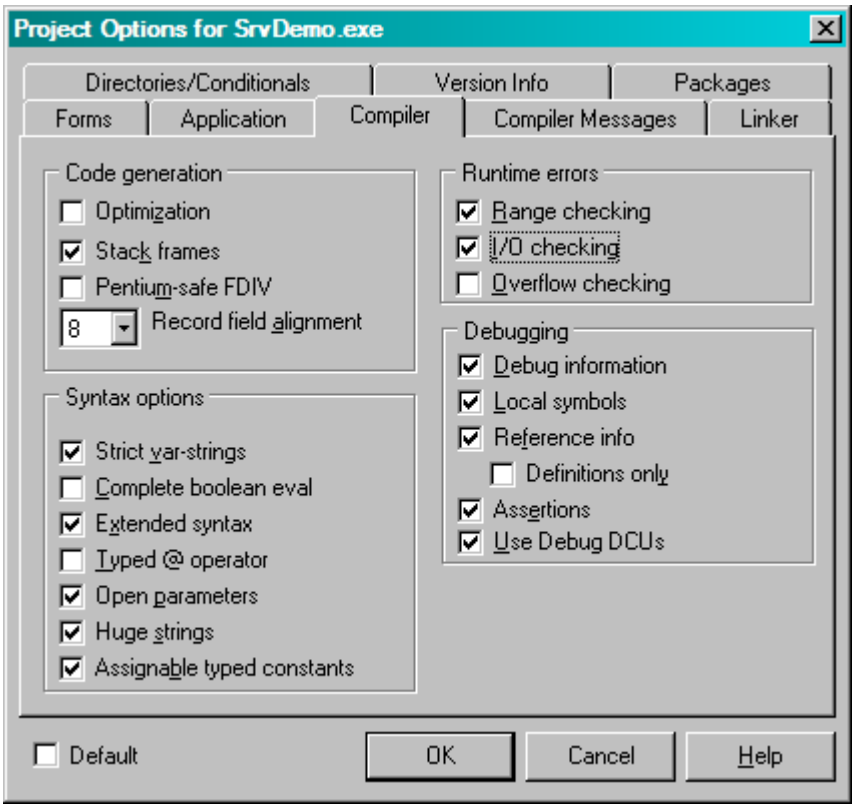
kbmMW debugging methods

Preparing a project for debugging

The following is shown for Delphi, but similar settings exist for Kylix and BCB.

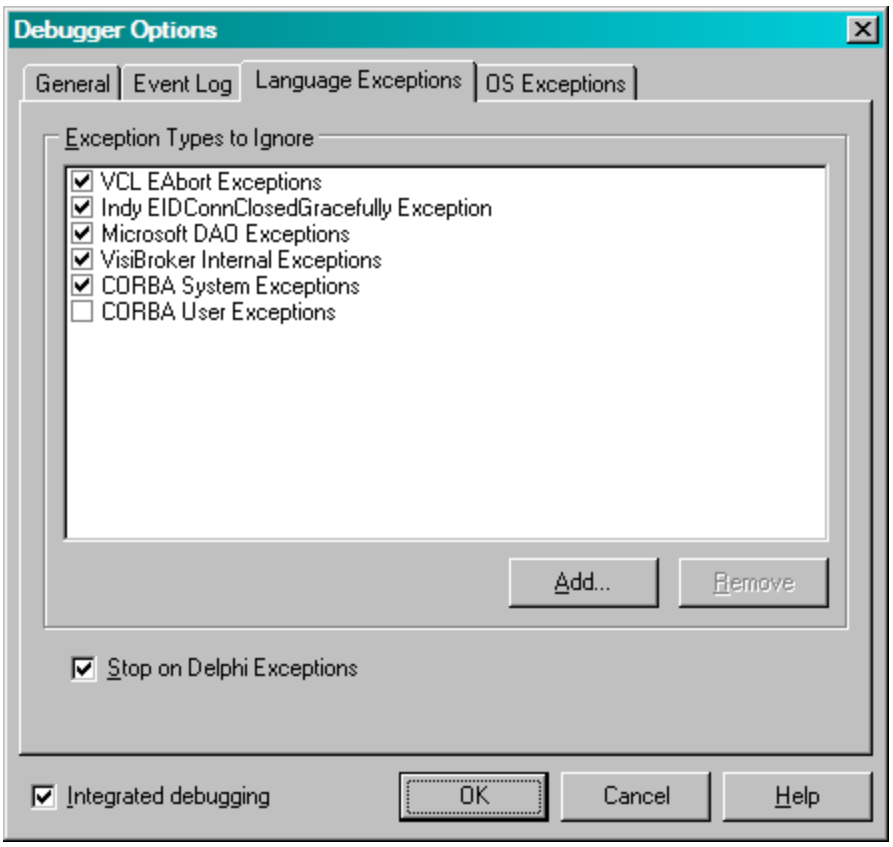
It's often best when debugging a problem in a kbmMW based setup, to start with running first the client, then the app. server in debug mode within the Delphi/Kylix/BCB IDE.

Debug mode means that you should rebuild your application (server and client/nodes) with debug turned on, no optimization, with stack frames etc. A good debug setting is shown here:
(Project/Options)



The Linker pane is not too important at this stage.
However make sure to put the path to the kbmMW source code into the Directories/Search path for debugging purposes only.

Also setup your IDE to stop on exceptions and to use the integrated debugger:
(Tools/Debugger options)



Then rebuild your project. It may take longer this time, as it will also rebuild all the kbmMW source files to include debug information.

Now run your client or application server within the IDE depending on which you suspect to have the problem at first.

Do the operations you have seen provoke the problem, and notice when Delphi stops on an exception.

Some exceptions are completely valid. For example, if you are using Indy transports, you will often get exceptions about client connected or disconnected on the server side. This is as designed in Indy, and can safely be ignored.

This way you may be able to see an exception being thrown at some point. If it's a kbmMW exception, it will show up with a general explanation of what went wrong. This may give you a clue as to what the problem is essentially about.

Invoking runtime trace on a kbmMW project

kbmMW have several nice tracing facilities which are built in but default dormant. The same facilities can also be manually utilized by the developer to add own additional debugging to a project.

Inside kbmMW several so called trace points have been embedded. They are default inactive and needs the developer to setup some special global variables to activate them. Once activated some other global variables control how the trace should look like and where to place the trace.

The following global variables should be setup in one place within the application to trace (client or app. server or node), typically in the OnCreate event of a central main form, or in the initialization section of a main unit.

kbmMWDebugLevel

Controls the verboseness of the trace. The more verbose, the more trace points are potentially activated with a higher level of details. The values can be:
mwdlNone,**mwdlBasic**,**mwdlAdvanced**,**mwdlAll**
Default is mwdlNone

kbmMWDebugTypes

Specifies what type of trace points should be considered for the trace. This and kbmMWDebugLevel is what actually controls which trace points go active. This is a set variable and thus multiple values can be specified to enable more types of trace.

mwdtTransport
If specified will enable tracing of all transport communication. For example on the server, one request from a client to a server would often result in multiple trace points being activated showing the communication when first entered from the client, after decryption, after uncompressing, after header information have been obtained etc. Each of the trace points often provide more and more detailed interpreted information if one of the text outputs have been chosen.

mwdtService
Can be used by developers to activate own trace points put into custom services. kbmMW may itself include trace points using of this type at some point.

mwdtServer
Can be used by developers to activate own trace points which are specifically server rather than service oriented. kbmMW may itself include trace points using of this type at some point.

mwdtDataset

If specified will enable tracing of all dataset related operations. Currently it primarily supports tracing resolve operations with a very high level of details. You will see which SQL has been generated and which parameters (field values) are being sent to the backend database. If you have problems resolving some specific set of data, try to enable this and it will often become apparent what was wrong and what resolver setting should be changed.

kbmMWDebugWhere

Controls where the trace should be placed. It can be set to one of the following values:

mwddDialog: Each time a trace point is reached, a dialog pops up with the interpreted trace information along with headers related to the type of trace. Be careful about using this, as you may end up having to click/press OK many times.

mwddDebug: Similar to mwddDialog, except that the trace is written to the event viewer.

mwddTextFile: Similar to mwddDialog, except that the trace is written to a new unique trace file.

mwddRawFile: Each time a trace point is reached, a new unique trace file is created and the raw buffer trace information for that trace point is written as a binary dump, to the file without any headers or any other additional information.

mwddRawTextFile: Each time a trace point is reached, a new trace file is created and the raw buffer trace information for that trace point is dumped as hexadecimal string values along with headers related to the type of trace.

mwddOneTextFile: Similar to mwddTextFile except that the trace is appended to one specific trace file which name and path is given by the global string variable kbmMWDebugFile.

mwddOneRawFile: Similar to mwddRawFile except that the trace is appended to one specific trace file the same way as mwddOneTextFile.

mwddOneRawTextFile: Similar to mwddRawTextFile except that the trace is appended to one specific trace file the same way as mwddOneTextFile.

kbmMWDebugFile

Is used when one of the mwddOne...File settings was chosen, to specify where the debug file should be placed, and what name it should have. For Windows, the name is default ‘c:\kbmMW_DebugDump.log' and for Linux its default '~/kbmMW_DebugDump.log'

If the mwdd...File settings is used, a unique new file is generated for each time an active trace point is reached. The file is placed in the users default temporary directory which for XP typically is

```
c:\document and settings\<username>\Local Settings\Temp
```

The name is formatted like this:

```
KBMMW_<type>_<applicationname>_<someuniquenumber>.tmp
```

<type> is the type of the trace point that produced this trace file.
<applicationname> is the name of the application that produced this trace file.
<someuniquenumber> is a several digits hexadecimal value that makes the filename unique.

Communication problems

Your client don't seem to be able to access the application server

Connection full (eg. TCPIP)

In a connection full transport setup (for example any of the TCPIP based transports) first look for the following things:

- Is the server actually running and listening on the expected port numbers?
  For most transports, there are port settings which must match in both ends to be able to establish a connection.

- Is the server's transport setup differently with regards to **StreamFormat** and **VerifyTransfer** than the client's transport?
  For most transports, these two properties must be the same on the client and the server transports. If they are not, the client may be able to connect, but the server wont understand what the client is requesting. As a result of that, the server usually kills the client connection, and the client will notice a 'connection lost'.

- Are there differences in the **Params** setting on the client and server transports?
  This may not be a crucial problem depending on the actual **StreamFormat** and transport type. Some **StreamFormat**'s like the HTTP one accept some extra settings for the client part, which are provided through the **Params** property. However other transports may require a setting specified on both sides.

Connection less (eg. UDP broadcast messaging)

In a connection less transport setup, clients (nodes) don't really connect to the application server (or other nodes), but instead simply live on the LAN listening for incoming messages. Only when there is something specific to send, the message will be put on the LAN, but usually without establishing a connection to any receiving point first.
Thus look for the following things if the expected receiving node does not get the data expected:

- Have you activated the client transport or set the server transport to listen?
  If the client or server transport is not active, messages will be queued up internally and only sent when the client transport is made active (Active:=true) and respectively when the server transports start to listen.

- Do all the nodes (clients and app. servers) have the same **ListenPort** and **SendPort** settings?
  If not, make sure to set all those nodes that are expected to communicate together to the same number.

- Do the nodes have **ListenIP** set to an IP mask that matches the senders IP?
  If not sure, try to change to 0.0.0.0 to listen for incoming messages originating from any IP address. If it works, you know where to tune.

- Do the nodes have **SendIP** set to an IP mask that match the node who you would like to see the sent message?
  If not sure, set it to 255.255.255.255 and check if it works now.

For both scenarios also check the subnet mask on the nodes/clients/servers. It can be displayed in most Windows installations by opening a MSDOS window, and type '**ipconfig'**.

It will give a overview of the current network adapters and who their settings are.
The subnet mask is used by the computers to decide what part of an IP address is a network identifier and what part is a host identifier. Specially for broadcast operations, its important that all participant nodes all belong to the same network.

Read more about how the IP address and subnet mask are used here:
http://www.burningvoid.com/iaq/network-class.html

Another important check is if encryption or compression has been enabled on any of the nodes. If it has, try to disable it and see if communication starts to work. Usually make sure that keys and chosen algorithms are the same on both sides. For SSL type security, the problems are usually to do with certificates not loaded correctly (the public part of a server certificate should be known by the client, and the client's public part should be known by the server, while the server's private key should be loaded by the server, and the client's private key should be loaded by the client.

Also check certificates expiration time, algorithms etc. Refer to the SSL whitepaper downloadable from the C4D site at www.components4developers.com.

Troubleshooting kbmMW exceptions

Here are a list of some of the exceptions that potentially can occur. The list is not complete, but most exceptions are self explaining, and usually either are because of lack of a property setting, or some version conflicts.

Duplicate state ID – client.
Should never be raised during normal operations. However if a client, utilizing stateful services, is loosing its server because of a server shutdown and the client do not itself clear all the state identifiers it holds on to, the server may at some point provide a state identifier which the client already have in its state identifier cache. This is an error and will raise the exception.

Transport must be set – client
Will be raised if the Transport property of a **TkbmMWCustomSimpleClient** or descendant is not pointing on a client transport when a request is sent using the SendRequest or SendRequestEx methods.

Wrong ... record version from server – client
May be raised if the client is using one version of kbmMW and the server another, and the client receives a response on a call to a query or file service which have a totally different record layout than expected.

Versioning not enabled. Resolving not possible – datasets
Will be raised if EnableVersioning is set to false on a dataset, and Resolve is being attempted.

TkbmMWServer does not have owner. Not able to determine connection pools. – server/stats
Will be raised if the TkbmMWServer component don't have an owner and statistics are enabled. In this case, the statistics module within kbmMW wont be able to produce connection pool related statistics.

No connection could be obtained – dataset or client
Could not obtain a best connection – dataset or client
Will be raised if either all connections within a connection pool is in use or it's not possible to queue to get one, or because the connection pool is not active.

Timeout/error waiting for connection – dataset or client
Will be raised if a request for a connection does not produce a connection within the given timeout.

Variable endmarker not found – dataset macro
Will be raised if a macro reference in a query/stored procedure name is not correctly formatted. The format is {%macroname%}

KeyFieldNames not specified. Resolving not possible – dataset resolving
Will be raised, if not minimum one field is specified in the dataset's KeyFieldNames property, before resolving.

TableName not specified. Resolving not possible – dataset resolving
Will be raised, if not minimum one table name is specified in the dataset's TableName property, before resolving.

Operation aborted – resolving and other places
If the developer raise the EkbmMWAbortException exception at some point, which for example happens during resolve if the Abort argument of the OnInsert, OnDelete and OnModify events of the resolver is set to true, this message will be shown.

Insert failed - resolving
Modify failed - resolving
Delete failed - resolving
Is raised when a resolve operation which should result in at least one record being affected on the database, do not affect any records. The operation depends on the database and the database API being able to return a value for the number of records affected by the latest operation. Some of the multi database API's do support this, but the database connected to might not. Usually when the exception is raised however, its truly valid in the sense that no records were affected by the operation.

Multiple resultset statements not allowed – dataset
Is raised if a multi statement query was provided and more than one statement within the multi statement query returns rows. Only one of the statements is allowed to return rows.

Variant type not convertible to a Java object. VarType= - Java service
Is raised if a variant argument is not possible to be converted to a Java type.
The supported variant types are: varBoolean, varShortInt, varByte, varDouble, varCurrency, varSingle, varSmallInt, varWord, varInteger, varInt64, varDate, varEmpty, varNull, varString, varArray.

Subject too long (>1000 chars) – messaging
Will be raised if the subject is more than 1000 characters for a messaging being published via a so called fragmenting transport. An example of a fragmenting transport is the UDP broadcast messaging transport which is able to take a large message and split it into multiple smaller packets. Due to the typical restrictions on a transport that needs fragmentation for larger messages, a limit of 1000 characters for a subject has been imposed to make room for some message data in each packet.

Supply of query statement not allowed. – remote dataset
Will be raised if the client is providing a query statement in the client's Query/SQL property and the server have the query service property AllowClientStatement set to false, or if the client is providing the name of a named query (@name) in the query statement and AllowClientNamedQuery is set to false.

Supply of execute statement for named query not allowed – remote dataset
Supply of query statement for named query not allowed – remote dataset
Will be raised if the client is providing a query statement (@name@statement) for a named query in the client's Query/SQL property and the server have the query service property AllowClientStatement set to false while having AllowClientNamedQuery set to true.

Supply of query keyfields not allowed – remote dataset resolve
Will be raised if the client provides anything in the client dataset's KeyFields property and the query service property AllowClientKeyFields is set to false.

Supply of query tablename not allowed – remote dataset resolve
Will be raised if the client provides anything in the client dataset's TableName property and the query service property AllowClientTableName is set to false.

ReadVariantArray only supports one dimension – transport
ReadVariantByteArray only supports one dimension – transport
WriteVariantArray writes arrays of one dimension only – transport
WriteVariantByteArray writes arrays of one dimension only - transport
Will be raised if transfer of a multi dimensional variant array is being attempted. Instead make it single dimension, and nest the other dimension in as single dimension arrays. Eg. an array [2,3]:
ABBB
ABBB

Object to stream must implement IkbmMWObject interface – remote able objects
Will be raised if the developer is trying to pass an object between two nodes/server/client, which do not implement the IkbmMWObject interface. Either implement it yourself, or descend from TkbmMWObject.

Missing Database template for ..... connection pool – dataset
Raised if the Database property of a database connection pool is not set.

DBISAM4 - ''resultset'' parameter not defined for stored procedure. PerformQuery aborted – DBISAM4 dataset
Raised if a stored procedure which do not return a resultset parameter is 'opened' rather than executed.

Ressource invalidated. Try later – file service
Raised if a client to server file upload has gone bad due to disk problems or similar, and a file client tries to access the file before its cleaned up by the server.

Ressource busy. Access denied. – file service
Raised if a client tries to access a file using a mode which is incompatible with the current mode of the file. For example if two clients try to write to the same file at the same time, or one is writing while the other is reading.

Ressource busy. Timeout waiting for access. – file service
Raised if a client tries to access a file resource, but is put on queue and has been waiting for more than 10 seconds.

Access denied. – file service
Raised if the file operation was not allowed by the operative system.

Too many files matching (>10000) – file service
Raised if a file listing has been started, and more than 10000 files are found matching the search.

Permission denied – file service
Raised if the file client operation was denied of security reasons by the file server. This is controlled in the OnFileAccess event of the servers file pool.

Recursive listing denied – file service
Raised if a recursive file listing was requested, but the server do not allow for recursive file listing operations (missing mwfapSubDirs flag)

Past end of file – file service
Raised if the file client is trying to obtain a file block which is after the end of the file.

Not supported on Linux – file service
Raised if a file client tries to set attributes and/or file times/dates on a file on Linux.

This concludes the whitepaper about debugging.

Kim Madsen
Components4Developers