Application Forward Compatibility to DAT 2.0

Arkady Kanevsky

Abstract

DAT 2.0 is the major version of the DAT specification, and hence does not preserve source level backward compatibility. This short report outlines the signature changes of existing DAT 1.x APIs in DAT 2.0 APIs and provides guidance on how to port an application to DAT 2.0. The document consists of two sections. The first one provides guidance for the uDAPL users, while the second does it for kDAPL users. While most of the changes are identical for both groups, some are not. It was deemed better to repeat the same changes in both parts so that Consumers can read only the section which is of interest to them.

It is possible to generate a scripting tool which will do 90% of the port automatically but the DAT Collaborative decided that it is outside the scope of its work. Open source reference implementation effort(s) may consider this task.

1 uDAPL

1.1 Structure sizes

Several structures (e.g. dat_dto_completion_event_data, dat_cr_arrival_event_data, dat_lmr_param, dat_rmr_param, dat_cno_param, dat_ep_param, dat_ep_state) changed in size as new fields were added. Recompiling will handle the forward compatibility issues arising from memory allocation for the structures. If an application used a relative position of a field in a structure rather than the field name, then the source changes required are a lot more elaborate and can not be handled in generic manner.

1.2 DAT_LMR_TRIPLET and DAT_RMR_TRIPLET

The formats of both triplets have been changed but the names of the individual members of the structures have not. Since *pad* should not have been used by the code no changes in the code are expected if the fields were accessed via names and not positions in the structures. Thus, a simple recompile will handle forward compatibility.

1.3 DAT_RMR_BIND

The *dat_rmr_bind* signature has been changed with the addition of two new fields, *lmr_handle* and *va_type*. For forward compatibility, the *va_type* value of DAT_VA_TYPE_VA should be used to represent an OS Virtual Address. There is no automatic fix for *lmr_handle*. Applications need to store the LMR handle returned by *dat_lmr_create* together with the LMR Context and pass it to the *dat_rmr_bind* function.

1.4 DAT_CR_REJECT

dat_cr_reject now supports the local Consumer's ability to pass private data to a remote Consumer. The signature of the function has changed and the size of the private data and the pointer to the start of the private data need to be passed in. The older applications can specify *private_data_size* of 0, and

private_data of NULL for forward compatibility. On the requesting side there was already support for private data in the connection event, therefore no changes are needed for forward compatibility.

1.5 DAT_CNO_QUERY

While the signature of the function has not changed, the format of the *dat_cno_param* structure which it returns did change, as well as the *dat_cno_param_mask*. For forward compatibility, if the call used the mask value of DAT_CNO_FIELD_AGENT it should be replaced by DAT_CNO_FIELD_PROXY. If code accessed *agent* it should be replaced by *proxy.agent*.

2 kDAPL

2.1 Structure sizes

Several structures (e.g. dat_dto_completion_event_data, dat_cr_arrival_event_data, dat_lmr_param, dat_rmr_param, dat_ep_param, dat_ep_state) changed in size as new fields were added. Recompiling will handle the forward compatibility issues arising from memory allocation for the structures. If an application used a relative position of a field in a structure rather than the field name, then the source changes required are a lot more elaborate and can not be handled in generic manner.

2.2 DAT_LMR_KCREATE

The signature of the *dat_lmr_kcreate* has been changed with an additional *va_type*. For the forward compatibility, the *va_type* value of DAT_VA_TYPE_VA should be used to represent an OS Virtual Address.

2.3 DAT_CR_REJECT

dat_cr_reject now supports the local Consumer's ability to pass private data to a remote Consumer. The signature of the function has changed and the size of the private data and the pointer to the start of the private data need to be passed in. The older applications can specify *private_data_size* of 0, and *private_data* of NULL for forward compatibility. On the requesting side there was already support for private data in the connection event, therefore no changes are needed for forward compatibility.

2.4 DAT_RMR_BIND

The *dat_rmr_bind* signature has been changed with the addition of two new fields, *lmr_handle* and *va_type*. For forward compatibility, the *va_type* value of DAT_VA_TYPE_VA should be used to represent an OS Virtual Address. There is no automatic fix for *lmr_handle*. Applications need to store the LMR handle returned by *dat_lmr_kcreate* together with the LMR Context and pass it to the *dat_rmr_bind* function.

2.5 DAT_LMR_TRIPLET and DAT_RMR_TRIPLET

The formats of both triplets have been changed but the names of the individual members of the structures have not. Since *pad* should not have been used by the code, no changes in the code are expected if the fields were accessed via names and not positions in the structures. Thus, a simple recompile will handle forward compatibility.

2.6 DAT_REGION_DESCRIPTION

There are multiple changes in *dat_region_decsription* which is used in *dat_lmr_kcreate* and *dat_lmr_query* as part of the *dat_lmr_param* structure. If an application used the *for_pa* field, then it should use *for_ia* for forward compatibility. If an application used the *for_pointer* field, then it should use *for_platform* for forward compatibility. If an application used the *for_array* field, then it should use *for_physical* for forward compatibility.

2.7 DAT_EVD_MODIFY_UPCALL

The *dat_evd_modify_upcall* signature has been changed with one addition field, *upcall_flag*. For forward compatibility, the *upcall_flag* value of DAT_UPCALL_INVOC_NEW should be used.