Using the GNI and DMAPP APIs

S–2446–3103
Changes to this Document

This version of S–2446–3103 supports the CLE 3.1.UP03 release.

S–2446–3103

Added information

- MemHndlQueryAttr on page 60
- Additional CDM Modes (see Parameters on page 39)
- DMAPP collective operations (see Collective Operations on page 139)
- Extended RMA attribute structure and supporting interface.
  - dmapp_pi_reg_type on page 144
  - dmapp_rma_atts_ext on page 148
  - dmapp_init_ext on page 151
  - dmapp_get_rma_atts_ext on page 153
  - dmapp_set_rma_atts_ext on page 155

Revised information

- gni_cq_entry on page 119
# Contents

<table>
<thead>
<tr>
<th>Introduction [1]</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Software Stack</td>
<td>28</td>
</tr>
</tbody>
</table>

## Part I: The GNI API

### About the GNI API [2] 33

2.1 Functional Overview ................................................. 33
  2.1.1 Establish Communication Domain ............................... 33
  2.1.2 Create Completion Queue (CQ) ................................ 34
  2.1.3 Register Memory .................................................. 34
  2.1.4 Create Logical Endpoints ...................................... 35
  2.1.5 Transfer Data ...................................................... 35
    2.1.5.1 Fast Memory Access (FMA) .................................. 35
    2.1.5.2 Block Transfer Engine (BTE) ............................... 36
  2.1.6 Process Completion Queue ...................................... 37
2.2 Restrictions ......................................................... 37
2.3 Compiling ............................................................. 37

### GNI API Reference [3] 39

3.1 Naming Conventions .................................................. 39
3.2 Communication Domain ............................................... 39
  3.2.1 CdmCreate .......................................................... 39
    3.2.1.1 Synopsis ......................................................... 39
    3.2.1.2 Parameters ...................................................... 39
    3.2.1.3 Return Codes .................................................. 41
  3.2.2 CdmDestroy ........................................................ 41
    3.2.2.1 Synopsis ......................................................... 41
    3.2.2.2 Parameters ...................................................... 41
    3.2.2.3 Return Codes .................................................. 41
  3.2.3 CdmGetNicAddress ............................................... 41
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.3.1 Synopsis</td>
<td>41</td>
</tr>
<tr>
<td>3.2.3.2 Parameters</td>
<td>42</td>
</tr>
<tr>
<td>3.2.3.3 Return Codes</td>
<td>42</td>
</tr>
<tr>
<td>3.2.4 CdmAttach</td>
<td>42</td>
</tr>
<tr>
<td>3.2.4.1 Synopsis</td>
<td>42</td>
</tr>
<tr>
<td>3.2.4.2 Parameters</td>
<td>42</td>
</tr>
<tr>
<td>3.2.4.3 Return Codes</td>
<td>43</td>
</tr>
<tr>
<td>3.2.5 GetVersion</td>
<td>43</td>
</tr>
<tr>
<td>3.2.5.1 Synopsis</td>
<td>43</td>
</tr>
<tr>
<td>3.2.5.2 Parameters</td>
<td>44</td>
</tr>
<tr>
<td>3.2.5.3 Return Codes</td>
<td>44</td>
</tr>
<tr>
<td>3.2.6 ConfigureNTT</td>
<td>44</td>
</tr>
<tr>
<td>3.2.6.1 Synopsis</td>
<td>44</td>
</tr>
<tr>
<td>3.2.6.2 Parameters</td>
<td>44</td>
</tr>
<tr>
<td>3.2.6.3 Return Codes</td>
<td>45</td>
</tr>
<tr>
<td>3.2.7 ConfigureJob</td>
<td>45</td>
</tr>
<tr>
<td>3.2.7.1 Synopsis</td>
<td>46</td>
</tr>
<tr>
<td>3.2.7.2 Parameters</td>
<td>46</td>
</tr>
<tr>
<td>3.2.7.3 Return Codes</td>
<td>47</td>
</tr>
<tr>
<td>3.2.8 ConfigureNTTandJob</td>
<td>47</td>
</tr>
<tr>
<td>3.2.8.1 Synopsis</td>
<td>48</td>
</tr>
<tr>
<td>3.2.8.2 Parameters</td>
<td>48</td>
</tr>
<tr>
<td>3.2.8.3 Return Codes</td>
<td>49</td>
</tr>
<tr>
<td>3.3 Completion Queue Management</td>
<td>49</td>
</tr>
<tr>
<td>3.3.1 CqCreate</td>
<td>49</td>
</tr>
<tr>
<td>3.3.1.1 Synopsis</td>
<td>50</td>
</tr>
<tr>
<td>3.3.1.2 Parameters</td>
<td>50</td>
</tr>
<tr>
<td>3.3.1.3 Return Codes</td>
<td>51</td>
</tr>
<tr>
<td>3.3.2 CqDestroy</td>
<td>51</td>
</tr>
<tr>
<td>3.3.2.1 Synopsis</td>
<td>51</td>
</tr>
<tr>
<td>3.3.2.2 Parameters</td>
<td>51</td>
</tr>
<tr>
<td>3.3.2.3 Return Codes</td>
<td>51</td>
</tr>
<tr>
<td>3.4 Memory Registration</td>
<td>52</td>
</tr>
<tr>
<td>3.4.1 MemRegister</td>
<td>52</td>
</tr>
<tr>
<td>3.4.1.1 Virtual Memory Domain Handles</td>
<td>53</td>
</tr>
<tr>
<td>3.4.1.2 Synopsis</td>
<td>53</td>
</tr>
<tr>
<td>3.4.1.3 Parameters</td>
<td>54</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.4.1.4 Return Codes</td>
<td>55</td>
</tr>
<tr>
<td>3.4.2 MemRegisterSegments</td>
<td>56</td>
</tr>
<tr>
<td>3.4.2.1 Synopsis</td>
<td>56</td>
</tr>
<tr>
<td>3.4.2.2 Parameters</td>
<td>57</td>
</tr>
<tr>
<td>3.4.2.3 Return Codes</td>
<td>58</td>
</tr>
<tr>
<td>3.4.3 SetMddResources</td>
<td>59</td>
</tr>
<tr>
<td>3.4.3.1 Synopsis</td>
<td>59</td>
</tr>
<tr>
<td>3.4.3.2 Parameters</td>
<td>59</td>
</tr>
<tr>
<td>3.4.3.3 Return Codes</td>
<td>59</td>
</tr>
<tr>
<td>3.4.4 MemDeregister</td>
<td>59</td>
</tr>
<tr>
<td>3.4.4.1 Synopsis</td>
<td>59</td>
</tr>
<tr>
<td>3.4.4.2 Parameters</td>
<td>60</td>
</tr>
<tr>
<td>3.4.4.3 Return Codes</td>
<td>60</td>
</tr>
<tr>
<td>3.4.5 MemHndlQueryAttr</td>
<td>60</td>
</tr>
<tr>
<td>3.4.5.1 Synopsis</td>
<td>60</td>
</tr>
<tr>
<td>3.4.5.2 Parameters</td>
<td>60</td>
</tr>
<tr>
<td>3.4.5.3 Return Codes</td>
<td>60</td>
</tr>
<tr>
<td>3.5 Logical Endpoint</td>
<td>61</td>
</tr>
<tr>
<td>3.5.1 EpCreate</td>
<td>61</td>
</tr>
<tr>
<td>3.5.1.1 Synopsis</td>
<td>61</td>
</tr>
<tr>
<td>3.5.1.2 Parameters</td>
<td>61</td>
</tr>
<tr>
<td>3.5.1.3 Return Codes</td>
<td>61</td>
</tr>
<tr>
<td>3.5.2 EpSetEventData</td>
<td>61</td>
</tr>
<tr>
<td>3.5.2.1 Synopsis</td>
<td>62</td>
</tr>
<tr>
<td>3.5.2.2 Parameters</td>
<td>62</td>
</tr>
<tr>
<td>3.5.2.3 Return Codes</td>
<td>62</td>
</tr>
<tr>
<td>3.5.3 EpBind</td>
<td>62</td>
</tr>
<tr>
<td>3.5.3.1 Synopsis</td>
<td>62</td>
</tr>
<tr>
<td>3.5.3.2 Parameters</td>
<td>63</td>
</tr>
<tr>
<td>3.5.3.3 Return Codes</td>
<td>63</td>
</tr>
<tr>
<td>3.5.4 EpUnbind</td>
<td>63</td>
</tr>
<tr>
<td>3.5.4.1 Synopsis</td>
<td>63</td>
</tr>
<tr>
<td>3.5.4.2 Parameters</td>
<td>63</td>
</tr>
<tr>
<td>3.5.4.3 Return Codes</td>
<td>64</td>
</tr>
<tr>
<td>3.5.5 EpDestroy</td>
<td>64</td>
</tr>
<tr>
<td>3.5.5.1 Synopsis</td>
<td>64</td>
</tr>
<tr>
<td>3.5.5.2 Parameters</td>
<td>64</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>3.5.15 PostDataProbeById</td>
<td>74</td>
</tr>
<tr>
<td>3.5.15.1 Synopsis</td>
<td>74</td>
</tr>
<tr>
<td>3.5.15.2 Parameters</td>
<td>75</td>
</tr>
<tr>
<td>3.5.15.3 Return Codes</td>
<td>75</td>
</tr>
<tr>
<td>3.5.16 PostDataProbeWaitById</td>
<td>75</td>
</tr>
<tr>
<td>3.5.16.1 Synopsis</td>
<td>75</td>
</tr>
<tr>
<td>3.5.16.2 Parameters</td>
<td>76</td>
</tr>
<tr>
<td>3.5.16.3 Return Codes</td>
<td>76</td>
</tr>
<tr>
<td>3.6 FMA DM</td>
<td>76</td>
</tr>
<tr>
<td>3.6.1 PostFma</td>
<td>76</td>
</tr>
<tr>
<td>3.6.1.1 Synopsis</td>
<td>76</td>
</tr>
<tr>
<td>3.6.1.2 Parameters</td>
<td>77</td>
</tr>
<tr>
<td>3.6.1.3 Return Codes</td>
<td>77</td>
</tr>
<tr>
<td>3.7 FMA Short Messaging (SMSG)</td>
<td>77</td>
</tr>
<tr>
<td>3.7.1 SmsgInit</td>
<td>77</td>
</tr>
<tr>
<td>3.7.1.1 Synopsis</td>
<td>77</td>
</tr>
<tr>
<td>3.7.1.2 Parameters</td>
<td>78</td>
</tr>
<tr>
<td>3.7.1.3 Return Codes</td>
<td>78</td>
</tr>
<tr>
<td>3.7.2 SmsgSend</td>
<td>78</td>
</tr>
<tr>
<td>3.7.2.1 Synopsis</td>
<td>79</td>
</tr>
<tr>
<td>3.7.2.2 Parameters</td>
<td>79</td>
</tr>
<tr>
<td>3.7.2.3 Return Codes</td>
<td>80</td>
</tr>
<tr>
<td>3.7.3 SmsgSendWTag</td>
<td>80</td>
</tr>
<tr>
<td>3.7.3.1 Synopsis</td>
<td>80</td>
</tr>
<tr>
<td>3.7.3.2 Parameters</td>
<td>81</td>
</tr>
<tr>
<td>3.7.3.3 Return Codes</td>
<td>81</td>
</tr>
<tr>
<td>3.7.4 SmsgGetNext</td>
<td>81</td>
</tr>
<tr>
<td>3.7.4.1 Synopsis</td>
<td>81</td>
</tr>
<tr>
<td>3.7.4.2 Parameters</td>
<td>82</td>
</tr>
<tr>
<td>3.7.4.3 Return Codes</td>
<td>82</td>
</tr>
<tr>
<td>3.7.5 SmsgGetNextWTag</td>
<td>82</td>
</tr>
<tr>
<td>3.7.5.1 Synopsis</td>
<td>82</td>
</tr>
<tr>
<td>3.7.5.2 Parameters</td>
<td>82</td>
</tr>
<tr>
<td>3.7.5.3 Return Codes</td>
<td>83</td>
</tr>
<tr>
<td>3.7.6 SmsgRelease</td>
<td>83</td>
</tr>
<tr>
<td>3.7.6.1 Synopsis</td>
<td>83</td>
</tr>
<tr>
<td>3.7.6.2 Parameters</td>
<td>83</td>
</tr>
</tbody>
</table>
Using the GNI and DMAPP APIs

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.7.6.3</td>
<td>Return Codes</td>
<td>83</td>
</tr>
<tr>
<td>3.7.7</td>
<td>SmsgSetMaxRetrans</td>
<td>84</td>
</tr>
<tr>
<td>3.7.7.1</td>
<td>Synopsis</td>
<td>84</td>
</tr>
<tr>
<td>3.7.7.2</td>
<td>Parameters</td>
<td>84</td>
</tr>
<tr>
<td>3.7.7.3</td>
<td>Return Codes</td>
<td>84</td>
</tr>
<tr>
<td>3.8</td>
<td>Shared Message Queue (MSGQ)</td>
<td>85</td>
</tr>
<tr>
<td>3.8.1</td>
<td>MsgqInit</td>
<td>85</td>
</tr>
<tr>
<td>3.8.1.1</td>
<td>Synopsis</td>
<td>86</td>
</tr>
<tr>
<td>3.8.1.2</td>
<td>Parameters</td>
<td>86</td>
</tr>
<tr>
<td>3.8.1.3</td>
<td>Return Codes</td>
<td>87</td>
</tr>
<tr>
<td>3.8.2</td>
<td>MsgqRelease</td>
<td>87</td>
</tr>
<tr>
<td>3.8.2.1</td>
<td>Synopsis</td>
<td>87</td>
</tr>
<tr>
<td>3.8.2.2</td>
<td>Parameters</td>
<td>87</td>
</tr>
<tr>
<td>3.8.2.3</td>
<td>Return Codes</td>
<td>88</td>
</tr>
<tr>
<td>3.8.3</td>
<td>MsgqGetConnAttrs</td>
<td>88</td>
</tr>
<tr>
<td>3.8.3.1</td>
<td>Synopsis</td>
<td>88</td>
</tr>
<tr>
<td>3.8.3.2</td>
<td>Parameters</td>
<td>88</td>
</tr>
<tr>
<td>3.8.3.3</td>
<td>Return Codes</td>
<td>89</td>
</tr>
<tr>
<td>3.8.4</td>
<td>MsgqConnect</td>
<td>89</td>
</tr>
<tr>
<td>3.8.4.1</td>
<td>Synopsis</td>
<td>89</td>
</tr>
<tr>
<td>3.8.4.2</td>
<td>Parameters</td>
<td>89</td>
</tr>
<tr>
<td>3.8.4.3</td>
<td>Return Codes</td>
<td>90</td>
</tr>
<tr>
<td>3.8.5</td>
<td>MsgqConnRelease</td>
<td>90</td>
</tr>
<tr>
<td>3.8.5.1</td>
<td>Synopsis</td>
<td>90</td>
</tr>
<tr>
<td>3.8.5.2</td>
<td>Parameters</td>
<td>90</td>
</tr>
<tr>
<td>3.8.5.3</td>
<td>Return Codes</td>
<td>91</td>
</tr>
<tr>
<td>3.8.6</td>
<td>MsgqSend</td>
<td>91</td>
</tr>
<tr>
<td>3.8.6.1</td>
<td>Synopsis</td>
<td>91</td>
</tr>
<tr>
<td>3.8.6.2</td>
<td>Parameters</td>
<td>92</td>
</tr>
<tr>
<td>3.8.6.3</td>
<td>Return Codes</td>
<td>92</td>
</tr>
<tr>
<td>3.8.7</td>
<td>MsgqSize</td>
<td>92</td>
</tr>
<tr>
<td>3.8.7.1</td>
<td>Synopsis</td>
<td>93</td>
</tr>
<tr>
<td>3.8.7.2</td>
<td>Parameters</td>
<td>93</td>
</tr>
<tr>
<td>3.8.7.3</td>
<td>Return Codes</td>
<td>93</td>
</tr>
<tr>
<td>3.8.8</td>
<td>MsgqProgress</td>
<td>93</td>
</tr>
<tr>
<td>3.8.8.1</td>
<td>Synopsis</td>
<td>93</td>
</tr>
<tr>
<td>3.8.8.2</td>
<td>Parameters</td>
<td>93</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>3.8.8.3 Return Codes</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>3.9 RDMA (BTE)</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>3.9.1 PostRdma</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>3.9.1.1 Synopsis</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>3.9.1.2 Parameters</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>3.9.1.3 Return Codes</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>3.10 Completion Queue Processing</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>3.10.1 CqTestEvent</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>3.10.1.1 Synopsis</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>3.10.1.2 Parameters</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>3.10.1.3 Return Codes</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>3.10.2 CqGetEvent</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>3.10.2.1 Synopsis</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>3.10.2.2 Parameters</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>3.10.2.3 Return Codes</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>3.10.3 CqWaitEvent</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>3.10.3.1 Synopsis</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>3.10.3.2 Parameters</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>3.10.3.3 Return Codes</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>3.10.4 CqVectorWaitEvent</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>3.10.4.1 Synopsis</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>3.10.4.2 Parameters</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>3.10.4.3 Return Codes</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>3.10.5 GetCompleted</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>3.10.5.1 Synopsis</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>3.10.5.2 Parameters</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>3.10.5.3 Return Codes</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>3.10.6 PostCqWrite</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>3.10.6.1 Synopsis</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>3.10.6.2 Parameters</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>3.10.6.3 Return Codes</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>3.10.7 CqErrorStr</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>3.10.7.1 Synopsis</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>3.10.7.2 Parameters</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>3.10.7.3 Return Codes</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>3.10.8 CqErrorRecoverable</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>3.10.8.1 Synopsis</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>3.10.8.2 Parameters</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>3.10.8.3 Return Codes</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>3.11 Error Handling</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>3.11.1 SubscribeErrors</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>3.11.1.1 Synopsis</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>3.11.1.2 Parameters</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>3.11.1.3 Return Codes</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>3.11.2 ReleaseErrors</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>3.11.2.1 Synopsis</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>3.11.2.2 Parameters</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>3.11.2.3 Return Codes</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>3.11.3 GetErrorMask</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>3.11.3.1 Synopsis</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>3.11.3.2 Parameters</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>3.11.3.3 Return Codes</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>3.11.4 SetErrorMask</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>3.11.4.1 Synopsis</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>3.11.4.2 Parameters</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>3.11.4.3 Return Codes</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>3.11.5 GetErrorEvent</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>3.11.5.1 Synopsis</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>3.11.5.2 Parameters</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>3.11.5.3 Return Codes</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>3.11.6 WaitErrorEvents</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>3.11.6.1 Synopsis</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>3.11.6.2 Parameters</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>3.11.6.3 Return Codes</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>3.11.7 SetErrorPtag</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>3.11.7.1 Synopsis</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>3.11.7.2 Parameters</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>3.11.7.3 Return Codes</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>3.12 Other</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>3.12.1 GetNumLocalDevices</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>3.12.1.1 Synopsis</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>3.12.1.2 Parameters</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>3.12.1.3 Return Codes</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>3.12.2 GetLocalDeviceIds</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>3.12.2.1</td>
<td>Synopsis</td>
<td>108</td>
</tr>
<tr>
<td>3.12.2.2</td>
<td>Parameters</td>
<td>108</td>
</tr>
<tr>
<td>3.12.2.3</td>
<td>Return Codes</td>
<td>108</td>
</tr>
<tr>
<td>3.13</td>
<td>Enumerations</td>
<td>108</td>
</tr>
<tr>
<td>3.13.1</td>
<td>gni_cq_mode</td>
<td>108</td>
</tr>
<tr>
<td>3.13.1.1</td>
<td>Synopsis</td>
<td>109</td>
</tr>
<tr>
<td>3.13.1.2</td>
<td>Constants</td>
<td>109</td>
</tr>
<tr>
<td>3.13.2</td>
<td>gni_fma_cmd_type</td>
<td>109</td>
</tr>
<tr>
<td>3.13.2.1</td>
<td>Synopsis</td>
<td>110</td>
</tr>
<tr>
<td>3.13.2.2</td>
<td>Constants</td>
<td>110</td>
</tr>
<tr>
<td>3.13.3</td>
<td>gni_mem_handle_attr</td>
<td>112</td>
</tr>
<tr>
<td>3.13.3.1</td>
<td>Synopsis</td>
<td>112</td>
</tr>
<tr>
<td>3.13.3.2</td>
<td>Constants</td>
<td>113</td>
</tr>
<tr>
<td>3.13.4</td>
<td>gni_post_state</td>
<td>113</td>
</tr>
<tr>
<td>3.13.4.1</td>
<td>Synopsis</td>
<td>113</td>
</tr>
<tr>
<td>3.13.4.2</td>
<td>Constants</td>
<td>114</td>
</tr>
<tr>
<td>3.13.5</td>
<td>gni_post_type</td>
<td>114</td>
</tr>
<tr>
<td>3.13.5.1</td>
<td>Synopsis</td>
<td>114</td>
</tr>
<tr>
<td>3.13.5.2</td>
<td>Constants</td>
<td>115</td>
</tr>
<tr>
<td>3.13.6</td>
<td>gni_return</td>
<td>115</td>
</tr>
<tr>
<td>3.13.6.1</td>
<td>Synopsis</td>
<td>115</td>
</tr>
<tr>
<td>3.13.6.2</td>
<td>Constants</td>
<td>115</td>
</tr>
<tr>
<td>3.13.7</td>
<td>gni_smsg_type</td>
<td>117</td>
</tr>
<tr>
<td>3.13.7.1</td>
<td>Synopsis</td>
<td>117</td>
</tr>
<tr>
<td>3.13.7.2</td>
<td>Constants</td>
<td>117</td>
</tr>
<tr>
<td>3.14</td>
<td>Structures</td>
<td>117</td>
</tr>
<tr>
<td>3.14.1</td>
<td>gni_error_event</td>
<td>117</td>
</tr>
<tr>
<td>3.14.1.1</td>
<td>Synopsis</td>
<td>117</td>
</tr>
<tr>
<td>3.14.1.2</td>
<td>Members</td>
<td>117</td>
</tr>
<tr>
<td>3.14.2</td>
<td>gni_error_mask</td>
<td>118</td>
</tr>
<tr>
<td>3.14.2.1</td>
<td>Synopsis</td>
<td>119</td>
</tr>
<tr>
<td>3.14.3</td>
<td>gni_cq_entry</td>
<td>119</td>
</tr>
<tr>
<td>3.14.3.1</td>
<td>Synopsis</td>
<td>119</td>
</tr>
<tr>
<td>3.14.3.2</td>
<td>Event types</td>
<td>119</td>
</tr>
<tr>
<td>3.14.3.3</td>
<td>User data</td>
<td>119</td>
</tr>
<tr>
<td>3.14.3.4</td>
<td>Status</td>
<td>120</td>
</tr>
<tr>
<td>3.14.4</td>
<td>gni_job_limits</td>
<td>120</td>
</tr>
</tbody>
</table>
Part II: The DMAPP API

About the DMAPP API [4]

4.1 DMAPP Programming Model ................................................................. 131
4.2 DMAPP Applications and Fork ............................................................... 132
4.3 DMAPP Applications and Threads ......................................................... 132
4.4 DMAPP Applications and File Descriptors ............................................ 132
4.5 DMAPP Application Intra-node Communication ...................................... 132
4.6 Compiling and Launching DMAPP Applications ...................................... 132
4.7 Resiliency ............................................................................................... 133
4.8 DMAPP Remote Memory Access ............................................................ 133
4.9 DMAPP API ............................................................................................ 135
   4.9.1 Initialization and Query Functions .................................................. 135
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.9.2 One-sided RMA Functions</td>
<td>135</td>
</tr>
<tr>
<td>4.9.2.1 Contiguous Functions</td>
<td>136</td>
</tr>
<tr>
<td>4.9.2.2 Strided Functions</td>
<td>136</td>
</tr>
<tr>
<td>4.9.2.3 Scatter/Gather Functions</td>
<td>137</td>
</tr>
<tr>
<td>4.9.2.4 PE-strided Functions</td>
<td>137</td>
</tr>
<tr>
<td>4.9.2.5 DMAPP AMO Functions</td>
<td>138</td>
</tr>
<tr>
<td>4.9.2.6 DMAPP Synchronization Functions</td>
<td>139</td>
</tr>
<tr>
<td>4.9.2.7 Collective Operations</td>
<td>139</td>
</tr>
<tr>
<td>4.9.3 Symmetric Heap Functions</td>
<td>141</td>
</tr>
<tr>
<td>4.9.4 Checkpoint Restart Functions</td>
<td>141</td>
</tr>
<tr>
<td>DMAPP API Reference [5]</td>
<td>143</td>
</tr>
<tr>
<td>5.1 DMAPP Enumerations</td>
<td>143</td>
</tr>
<tr>
<td>5.1.1 dmapp_type</td>
<td>143</td>
</tr>
<tr>
<td>5.1.1.1 Synopsis</td>
<td>143</td>
</tr>
<tr>
<td>5.1.1.2 Constants</td>
<td>143</td>
</tr>
<tr>
<td>5.1.2 dmapp_routing_type</td>
<td>143</td>
</tr>
<tr>
<td>5.1.3 dmapp_pi_reg_type</td>
<td>144</td>
</tr>
<tr>
<td>5.1.4 dmapp_c_op</td>
<td>144</td>
</tr>
<tr>
<td>5.1.5 dmapp_c_type</td>
<td>144</td>
</tr>
<tr>
<td>5.2 DMAPP Structures</td>
<td>145</td>
</tr>
<tr>
<td>5.2.1 dmapp_seg_desc</td>
<td>145</td>
</tr>
<tr>
<td>5.2.1.1 Synopsis</td>
<td>145</td>
</tr>
<tr>
<td>5.2.1.2 Members</td>
<td>145</td>
</tr>
<tr>
<td>5.2.2 dmapp_jobinfo</td>
<td>145</td>
</tr>
<tr>
<td>5.2.2.1 Synopsis</td>
<td>146</td>
</tr>
<tr>
<td>5.2.2.2 Members</td>
<td>146</td>
</tr>
<tr>
<td>5.2.3 dmapp_rma_attrs</td>
<td>146</td>
</tr>
<tr>
<td>5.2.3.1 Synopsis</td>
<td>146</td>
</tr>
<tr>
<td>5.2.3.2 Members</td>
<td>146</td>
</tr>
<tr>
<td>5.2.4 dmapp_rma_attrs_ext</td>
<td>148</td>
</tr>
<tr>
<td>5.2.4.1 Synopsis</td>
<td>148</td>
</tr>
<tr>
<td>5.2.4.2 Members</td>
<td>148</td>
</tr>
<tr>
<td>5.2.5 dmapp_syncid</td>
<td>149</td>
</tr>
<tr>
<td>5.2.5.1 Synopsis</td>
<td>149</td>
</tr>
<tr>
<td>5.2.6 dmapp_c_pset_desc_t</td>
<td>150</td>
</tr>
<tr>
<td>5.2.6.1 Synopsis</td>
<td>150</td>
</tr>
<tr>
<td>5.2.7 dmapp_c_pset_handle_t</td>
<td>150</td>
</tr>
</tbody>
</table>
5.2.7.1 Synopsis ............................................. 150

5.3 DMAPP Functions ........................................... 151

5.3.1 dmapp_init ............................................. 151
   5.3.1.1 Synopsis ........................................... 151
   5.3.1.2 Parameters ......................................... 151
   5.3.1.3 Return Codes ...................................... 151

5.3.2 dmapp_init_ext .......................................... 151
   5.3.2.1 Synopsis ........................................... 152
   5.3.2.2 Parameters ......................................... 152
   5.3.2.3 Return Codes ...................................... 152

5.3.3 dmapp_finalize ........................................... 152
   5.3.3.1 Synopsis ........................................... 152
   5.3.3.2 Return Codes ...................................... 152

5.3.4 dmapp_get_jobinfo ....................................... 152
   5.3.4.1 Synopsis ........................................... 153
   5.3.4.2 Parameters ......................................... 153
   5.3.4.3 Return Codes ...................................... 153

5.3.5 dmapp_get_rma_attrs .................................... 153
   5.3.5.1 Synopsis ........................................... 153
   5.3.5.2 Parameters ......................................... 153
   5.3.5.3 Return Codes ...................................... 153

5.3.6 dmapp_get_rma_attrs_ext .................................. 153
   5.3.6.1 Synopsis ........................................... 154
   5.3.6.2 Parameters ......................................... 154
   5.3.6.3 Return Codes ...................................... 154

5.3.7 dmapp_set_rma_attrs .................................... 154
   5.3.7.1 Synopsis ........................................... 154
   5.3.7.2 Parameters ......................................... 154
   5.3.7.3 Return Codes ...................................... 155

5.3.8 dmapp_set_rma_attrs_ext .................................. 155
   5.3.8.1 Synopsis ........................................... 155
   5.3.8.2 Parameters ......................................... 155
   5.3.8.3 Return Codes ...................................... 155

5.3.9 dmapp_put_nb ............................................ 155
   5.3.9.1 Synopsis ........................................... 155
   5.3.9.2 Parameters ......................................... 156
   5.3.9.3 Return Codes ...................................... 156
5.3.28.2 Parameters ........................................ 180
5.3.28.3 Return Codes ...................................... 180
5.3.29 dmapp_put_ixpe ........................................ 181
  5.3.29.1 Synopsis ........................................ 181
  5.3.29.2 Parameters ........................................ 181
  5.3.29.3 Return Codes ...................................... 182
5.3.30 dmapp_scatter_ixpe_nb ............................... 182
  5.3.30.1 Synopsis ........................................ 182
  5.3.30.2 Parameters ........................................ 183
  5.3.30.3 Return Codes ...................................... 183
5.3.31 dmapp_scatter_ixpe_nbi .............................. 183
  5.3.31.1 Synopsis ........................................ 184
  5.3.31.2 Parameters ........................................ 184
  5.3.31.3 Return Codes ...................................... 184
5.3.32 dmapp_scatter_ixpe ................................... 185
  5.3.32.1 Synopsis ........................................ 185
  5.3.32.2 Parameters ........................................ 185
  5.3.32.3 Return Codes ...................................... 186
5.3.33 dmapp_gather_ixpe_nb ............................... 186
  5.3.33.1 Synopsis ........................................ 186
  5.3.33.2 Parameters ........................................ 187
  5.3.33.3 Return Codes ...................................... 187
5.3.34 dmapp_gather_ixpe_nbi .............................. 187
  5.3.34.1 Synopsis ........................................ 188
  5.3.34.2 Parameters ........................................ 188
  5.3.34.3 Return Codes ...................................... 188
5.3.35 dmapp_gather_ixpe ................................... 189
  5.3.35.1 Synopsis ........................................ 189
  5.3.35.2 Parameters ........................................ 189
  5.3.35.3 Return Codes ...................................... 190
5.3.36 dmapp_aadd_qw_nb ................................. 190
  5.3.36.1 Synopsis ........................................ 190
  5.3.36.2 Parameters ........................................ 190
  5.3.36.3 Return Codes ...................................... 191
5.3.37 dmapp_aadd_qw_nbi ................................. 191
  5.3.37.1 Synopsis ........................................ 191
  5.3.37.2 Parameters ........................................ 191
<table>
<thead>
<tr>
<th>Section</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3.37.3</td>
<td>Return Codes</td>
<td>192</td>
</tr>
<tr>
<td>5.3.38</td>
<td><code>dmapp_aadd_qw</code></td>
<td>192</td>
</tr>
<tr>
<td>5.3.38.1</td>
<td>Synopsis</td>
<td>192</td>
</tr>
<tr>
<td>5.3.38.2</td>
<td>Parameters</td>
<td>192</td>
</tr>
<tr>
<td>5.3.38.3</td>
<td>Return Codes</td>
<td>193</td>
</tr>
<tr>
<td>5.3.39</td>
<td><code>dmapp_aadd_qw_nb</code></td>
<td>193</td>
</tr>
<tr>
<td>5.3.39.1</td>
<td>Synopsis</td>
<td>193</td>
</tr>
<tr>
<td>5.3.39.2</td>
<td>Parameters</td>
<td>193</td>
</tr>
<tr>
<td>5.3.39.3</td>
<td>Return Codes</td>
<td>194</td>
</tr>
<tr>
<td>5.3.40</td>
<td><code>dmapp_aadd_qw_nbi</code></td>
<td>194</td>
</tr>
<tr>
<td>5.3.40.1</td>
<td>Synopsis</td>
<td>194</td>
</tr>
<tr>
<td>5.3.40.2</td>
<td>Parameters</td>
<td>194</td>
</tr>
<tr>
<td>5.3.40.3</td>
<td>Return Codes</td>
<td>195</td>
</tr>
<tr>
<td>5.3.41</td>
<td><code>dmapp_aadd_qw</code></td>
<td>195</td>
</tr>
<tr>
<td>5.3.41.1</td>
<td>Synopsis</td>
<td>195</td>
</tr>
<tr>
<td>5.3.41.2</td>
<td>Parameters</td>
<td>195</td>
</tr>
<tr>
<td>5.3.41.3</td>
<td>Return Codes</td>
<td>196</td>
</tr>
<tr>
<td>5.3.42</td>
<td><code>dmapp_aor_qw_nb</code></td>
<td>196</td>
</tr>
<tr>
<td>5.3.42.1</td>
<td>Synopsis</td>
<td>196</td>
</tr>
<tr>
<td>5.3.42.2</td>
<td>Parameter</td>
<td>196</td>
</tr>
<tr>
<td>5.3.42.3</td>
<td>Return Codes</td>
<td>197</td>
</tr>
<tr>
<td>5.3.43</td>
<td><code>dmapp_aor_qw_nbi</code></td>
<td>197</td>
</tr>
<tr>
<td>5.3.43.1</td>
<td>Synopsis</td>
<td>197</td>
</tr>
<tr>
<td>5.3.43.2</td>
<td>Parameter</td>
<td>197</td>
</tr>
<tr>
<td>5.3.43.3</td>
<td>Return Codes</td>
<td>198</td>
</tr>
<tr>
<td>5.3.44</td>
<td><code>dmapp_aor_qw</code></td>
<td>198</td>
</tr>
<tr>
<td>5.3.44.1</td>
<td>Synopsis</td>
<td>198</td>
</tr>
<tr>
<td>5.3.44.2</td>
<td>Parameters</td>
<td>198</td>
</tr>
<tr>
<td>5.3.44.3</td>
<td>Return Codes</td>
<td>199</td>
</tr>
<tr>
<td>5.3.45</td>
<td><code>dmapp_axor_qw_nb</code></td>
<td>199</td>
</tr>
<tr>
<td>5.3.45.1</td>
<td>Synopsis</td>
<td>199</td>
</tr>
<tr>
<td>5.3.45.2</td>
<td>Parameters</td>
<td>199</td>
</tr>
<tr>
<td>5.3.45.3</td>
<td>Return Codes</td>
<td>200</td>
</tr>
<tr>
<td>5.3.46</td>
<td><code>dmapp_axor_qw_nbi</code></td>
<td>200</td>
</tr>
<tr>
<td>5.3.46.1</td>
<td>Synopsis</td>
<td>200</td>
</tr>
<tr>
<td>5.3.46.2</td>
<td>Parameter</td>
<td>200</td>
</tr>
<tr>
<td>5.3.46.3</td>
<td>Return Codes</td>
<td>201</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>5.3.47</td>
<td>dmapp_axor_qw</td>
<td>201</td>
</tr>
<tr>
<td>5.3.47.1</td>
<td>Synopsis</td>
<td>201</td>
</tr>
<tr>
<td>5.3.47.2</td>
<td>Parameters</td>
<td>201</td>
</tr>
<tr>
<td>5.3.47.3</td>
<td>Return Codes</td>
<td>202</td>
</tr>
<tr>
<td>5.3.48</td>
<td>dmapp_afadd_qw_nb</td>
<td>202</td>
</tr>
<tr>
<td>5.3.48.1</td>
<td>Synopsis</td>
<td>202</td>
</tr>
<tr>
<td>5.3.48.2</td>
<td>Parameters</td>
<td>203</td>
</tr>
<tr>
<td>5.3.48.3</td>
<td>Return Codes</td>
<td>203</td>
</tr>
<tr>
<td>5.3.49</td>
<td>dmapp_afadd_qw_nbi</td>
<td>203</td>
</tr>
<tr>
<td>5.3.49.1</td>
<td>Synopsis</td>
<td>203</td>
</tr>
<tr>
<td>5.3.49.2</td>
<td>Parameters</td>
<td>204</td>
</tr>
<tr>
<td>5.3.49.3</td>
<td>Return Codes</td>
<td>204</td>
</tr>
<tr>
<td>5.3.50</td>
<td>dmapp_afadd_qw</td>
<td>204</td>
</tr>
<tr>
<td>5.3.50.1</td>
<td>Synopsis</td>
<td>204</td>
</tr>
<tr>
<td>5.3.50.2</td>
<td>Parameters</td>
<td>205</td>
</tr>
<tr>
<td>5.3.50.3</td>
<td>Return Codes</td>
<td>205</td>
</tr>
<tr>
<td>5.3.51</td>
<td>dmapp_afand_qw_nb</td>
<td>205</td>
</tr>
<tr>
<td>5.3.51.1</td>
<td>Synopsis</td>
<td>205</td>
</tr>
<tr>
<td>5.3.51.2</td>
<td>Parameters</td>
<td>206</td>
</tr>
<tr>
<td>5.3.51.3</td>
<td>Return Codes</td>
<td>206</td>
</tr>
<tr>
<td>5.3.52</td>
<td>dmapp_afand_qw_nbi</td>
<td>206</td>
</tr>
<tr>
<td>5.3.52.1</td>
<td>Synopsis</td>
<td>206</td>
</tr>
<tr>
<td>5.3.52.2</td>
<td>Parameters</td>
<td>207</td>
</tr>
<tr>
<td>5.3.52.3</td>
<td>Return Codes</td>
<td>207</td>
</tr>
<tr>
<td>5.3.53</td>
<td>dmapp_afand_qw</td>
<td>207</td>
</tr>
<tr>
<td>5.3.53.1</td>
<td>Synopsis</td>
<td>207</td>
</tr>
<tr>
<td>5.3.53.2</td>
<td>Parameters</td>
<td>208</td>
</tr>
<tr>
<td>5.3.53.3</td>
<td>Return Codes</td>
<td>208</td>
</tr>
<tr>
<td>5.3.54</td>
<td>dmapp_axor_qw_nb</td>
<td>208</td>
</tr>
<tr>
<td>5.3.54.1</td>
<td>Synopsis</td>
<td>208</td>
</tr>
<tr>
<td>5.3.54.2</td>
<td>Parameters</td>
<td>209</td>
</tr>
<tr>
<td>5.3.54.3</td>
<td>Return Codes</td>
<td>209</td>
</tr>
<tr>
<td>5.3.55</td>
<td>dmapp_axor_qw_nbi</td>
<td>209</td>
</tr>
<tr>
<td>5.3.55.1</td>
<td>Synopsis</td>
<td>209</td>
</tr>
<tr>
<td>5.3.55.2</td>
<td>Parameters</td>
<td>210</td>
</tr>
<tr>
<td>5.3.55.3</td>
<td>Return Codes</td>
<td>210</td>
</tr>
<tr>
<td>5.3.56</td>
<td>dmapp_axor_qw</td>
<td>210</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.3.56</td>
<td>Synopsis</td>
<td>210</td>
</tr>
<tr>
<td>5.3.56.1</td>
<td>Parameters</td>
<td>211</td>
</tr>
<tr>
<td>5.3.56.2</td>
<td>Return Codes</td>
<td>211</td>
</tr>
<tr>
<td>5.3.57</td>
<td>dmapp_afor_qw_nb</td>
<td>211</td>
</tr>
<tr>
<td>5.3.57.1</td>
<td>Synopsis</td>
<td>211</td>
</tr>
<tr>
<td>5.3.57.2</td>
<td>Parameters</td>
<td>212</td>
</tr>
<tr>
<td>5.3.57.3</td>
<td>Return Codes</td>
<td>212</td>
</tr>
<tr>
<td>5.3.58</td>
<td>dmapp_afor_qw_nbi</td>
<td>212</td>
</tr>
<tr>
<td>5.3.58.1</td>
<td>Synopsis</td>
<td>212</td>
</tr>
<tr>
<td>5.3.58.2</td>
<td>Parameters</td>
<td>213</td>
</tr>
<tr>
<td>5.3.58.3</td>
<td>Return Codes</td>
<td>213</td>
</tr>
<tr>
<td>5.3.59</td>
<td>dmapp_afor_qw</td>
<td>213</td>
</tr>
<tr>
<td>5.3.59.1</td>
<td>Synopsis</td>
<td>213</td>
</tr>
<tr>
<td>5.3.59.2</td>
<td>Parameters</td>
<td>214</td>
</tr>
<tr>
<td>5.3.59.3</td>
<td>Return Codes</td>
<td>214</td>
</tr>
<tr>
<td>5.3.60</td>
<td>dmapp_afax_qw_nb</td>
<td>214</td>
</tr>
<tr>
<td>5.3.60.1</td>
<td>Synopsis</td>
<td>215</td>
</tr>
<tr>
<td>5.3.60.2</td>
<td>Parameters</td>
<td>215</td>
</tr>
<tr>
<td>5.3.60.3</td>
<td>Return Codes</td>
<td>215</td>
</tr>
<tr>
<td>5.3.61</td>
<td>dmapp_afax_qw_nbi</td>
<td>216</td>
</tr>
<tr>
<td>5.3.61.1</td>
<td>Synopsis</td>
<td>216</td>
</tr>
<tr>
<td>5.3.61.2</td>
<td>Parameters</td>
<td>216</td>
</tr>
<tr>
<td>5.3.61.3</td>
<td>Return Codes</td>
<td>216</td>
</tr>
<tr>
<td>5.3.62</td>
<td>dmapp_afax_qw</td>
<td>217</td>
</tr>
<tr>
<td>5.3.62.1</td>
<td>Synopsis</td>
<td>217</td>
</tr>
<tr>
<td>5.3.62.2</td>
<td>Parameters</td>
<td>217</td>
</tr>
<tr>
<td>5.3.62.3</td>
<td>Return Codes</td>
<td>217</td>
</tr>
<tr>
<td>5.3.63</td>
<td>dmapp_acswap_qw_nb</td>
<td>218</td>
</tr>
<tr>
<td>5.3.63.1</td>
<td>Synopsis</td>
<td>218</td>
</tr>
<tr>
<td>5.3.63.2</td>
<td>Parameters</td>
<td>218</td>
</tr>
<tr>
<td>5.3.63.3</td>
<td>Return Codes</td>
<td>219</td>
</tr>
<tr>
<td>5.3.64</td>
<td>dmapp_acswap_qw_nbi</td>
<td>219</td>
</tr>
<tr>
<td>5.3.64.1</td>
<td>Synopsis</td>
<td>219</td>
</tr>
<tr>
<td>5.3.64.2</td>
<td>Parameters</td>
<td>219</td>
</tr>
<tr>
<td>5.3.64.3</td>
<td>Return Codes</td>
<td>220</td>
</tr>
<tr>
<td>5.3.65</td>
<td>dmapp_acswap_qw</td>
<td>220</td>
</tr>
<tr>
<td>5.3.65.1</td>
<td>Synopsis</td>
<td>220</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>5.3.65.2 Parameters</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>5.3.65.3 Return Codes</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>5.3.66 dmapp_syncid_test</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>5.3.66.1 Synopsis</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>5.3.66.2 Parameters</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>5.3.66.3 Return Codes</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>5.3.67 dmapp_syncid_wait</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>5.3.67.1 Synopsis</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>5.3.67.2 Parameters</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>5.3.67.3 Return Codes</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>5.3.68 dmapp_gsync_test</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>5.3.68.1 Synopsis</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>5.3.68.2 Parameters</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>5.3.68.3 Return Codes</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>5.3.69 dmapp_gsync_wait</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>5.3.69.1 Synopsis</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>5.3.69.2 Return Codes</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>5.3.70 dmapp_c_pset_create</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>5.3.70.1 Synopsis</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>5.3.70.2 Parameters</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>5.3.70.3 Return Codes</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>5.3.71 dmapp_c_pset_export</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>5.3.71.1 Synopsis</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>5.3.71.2 Parameters</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>5.3.71.3 Return Codes</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>5.3.72 dmapp_c_barrier_join</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>5.3.72.1 Synopsis</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>5.3.72.2 Parameters</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>5.3.72.3 Return Codes</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>5.3.73 dmapp_c_pset_cancel_op</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>5.3.73.1 Synopsis</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>5.3.73.2 Parameters</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>5.3.73.3 Return Codes</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>5.3.74 dmapp_c_pset_destroy</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>5.3.74.1 Synopsis</td>
<td>227</td>
<td></td>
</tr>
<tr>
<td>5.3.74.2 Parameters</td>
<td>227</td>
<td></td>
</tr>
<tr>
<td>5.3.74.3 Return Codes</td>
<td>227</td>
<td></td>
</tr>
</tbody>
</table>
5.3.75 dmapp_c_greduce_start
  5.3.75.1 Synopsis
  5.3.75.2 Parameters
  5.3.75.3 Return Codes
5.3.76 dmapp_c_greduce_nelems_max
  5.3.76.1 Synopsis
  5.3.76.2 Parameters
  5.3.76.3 Return Codes
5.3.77 dmapp_c_pset_test
  5.3.77.1 Synopsis
  5.3.77.2 Parameters
  5.3.77.3 Return Codes
5.3.78 dmapp_c_pset_wait
  5.3.78.1 Synopsis
  5.3.78.2 Parameters
  5.3.78.3 Return Codes
5.3.79 dmapp_sheap_malloc
  5.3.79.1 Synopsis
  5.3.79.2 Parameters
5.3.80 dmapp_sheap_realloc
  5.3.80.1 Synopsis
  5.3.80.2 Parameters
5.3.81 dmapp_sheap_free
  5.3.81.1 Synopsis
  5.3.81.2 Parameters
5.3.82 dmapp_mem_register
  5.3.82.1 Synopsis
  5.3.82.2 Parameters
  5.3.82.3 Return Codes
5.3.83 dmapp_mem_unregister
  5.3.83.1 Synopsis
  5.3.83.2 Parameters
  5.3.83.3 Return Codes
5.3.84 dmapp_segdesc_compare
  5.3.84.1 Synopsis
  5.3.84.2 Parameters
  5.3.84.3 Return Codes
This guide includes reference information for the Generic Network Interface (GNI) and Distributed Shared Memory Application (DMAPP) APIs. The intended audience are programmers who are developing system software such as Partitioned Global Address Space (PGAS) compilers and communication libraries that use the Gemini based system interconnection network to transfer data between processors on a Cray XE system.

The Gemini application-specific integrated circuit (ASIC) provides an interface between the processors and the interconnection network. The ASIC provides the address translation mechanism, communication modes, and low-latency synchronization necessary to support the abstraction of a global, shared address space across the entire machine.

Each ASIC includes two network interface controllers (NICs), and an embedded interconnection switch (router). Each NIC is an independent, addressable endpoint in the network, therefore a single ASIC supports two nodes.

The Gemini based system interconnection network and its associated software provides the following features:

- Support for message passing, one-sided operations, and global address space programming models.
- Global synchronization. Global timing information is passed through the high-speed network to synchronize the scheduler interrupts and time-of-day clocks in all the processors.
- Gather/scatter performance. A symmetric address translation mode allows access to all nodes in a job without needing to modify the fast memory access (FMA) window. A windowing mechanism allows for processors with limited physical addressing capabilities to efficiently access remote memory. Network packet overhead is reduced so that network efficiency is high during these operations.
- Flat collectives. Support for atomic memory operations plus efficient scatters allows collectives to be programmed in a vector-like manner to scale much better than typical message-based algorithms.
- End-to-end data protection. Hardware support is provided so that all packets between the sender and receiver receive a cyclic redundancy check (CRC) to detect data corruption. Further, link-level data is resent if an error occurs while data is transiting a link.
Using the GNI and DMAPP APIs

- Network routing allows you to add and delete nodes from the network while it is running.
- Adaptive routing may be used for most network data, reducing sensitivity to network hot spots.

1.1 Software Stack

uGNI and DMAPP provide low-level communication services to user-space software. uGNI directly exposes the communications capabilities of the Gemini ASIC, and is extensively described in Part I, The GNI API. DMAPP implements a logically shared, distributed memory (DM) programming model, and is extensively described in Part II, The DMAPP API.

The uGNI and DMAPP APIs allow system software to realize as much of the hardware performance of the Gemini network ASIC as possible while being reasonably portable to its successors.
kGNI is a kernel module that presents to kernel-space code an API similar to that of uGNI. The GNI Core provides low-level services to both uGNI and kGNI. kGNI and GNI Core are both in the kGNI module.

The Generic Hardware Abstraction Layer (GHAL) isolates all software from the hardware specifics of the Cray network application-specific integrated circuit (ASIC). These components are not described further in this book.
Layered on top of uGNI and DMAPP are portable communication libraries (such as MPICH2 and Cray SHMEM) and the Partitioned Global Address Space (PGAS) compilers (such as UPC and Coarray Fortran, labeled F08, in Figure 1). These software components are extensively described in other books available from Cray Inc.

uGNI and DMAPP are packaged as libraries available with the Cray Linux Environment (CLE) 3.1 release and are installed in /opt/cray/ugni and /opt/cray/dmapp.
Part I: The GNI API
The GNI API includes two sets of function calls. User-level high-performance applications use uGNI functions while kernel-level drivers use kGNI functions. This chapter describes the functionality of the uGNI set of function calls, focusing on their direct interaction with the NIC.

2.1 Functional Overview

A high-performance user-level application would use the uGNI API to accomplish the following tasks in order to establish communication among its instances:

- Establish a communication domain and attach it to an NIC device
- Create one or more completion queues (CQs)
- Register memory for use by the Gemini network ASIC
- Create logical endpoints
- Use Fast Memory Access (FMA) or Remote Direct Memory Access (RDMA) to communicate between endpoints.
- Deregister memory to free up resource

Each of these tasks comprises a category of API functions as described in the following sections.

2.1.1 Establish Communication Domain

A Communication Domain is a software construct which defines a group of endpoints which can intercommunicate. The domain creation step establishes a unique set of domain properties including a unique identifier, which is used by the application to reference a particular instance of a communication domain.

The communication domain allows an application to enforce a protection scheme across all of its network transactions. The application attaches the domain to a specific NIC device.
Logical Endpoints are created within the communication domain and represent a virtual interface into the network. Communication takes place between endpoints on local and remote peers, where each endpoint is bound tightly to exactly one other endpoint. Logical endpoints may be used for initiating transactions. An application posts transaction requests to an endpoint to invoke communication through that endpoint. See Communication Domain on page 39.

### 2.1.2 Create Completion Queue (CQ)

Completion Queues (CQ) provide an event notification mechanism. For example, an application may use them to track the progress of Fast Memory Access (FMA) or Block Transfer Engine (BTE) requests, or to notify a remote node that data has been delivered to local memory.

An application must first initialize a CQ to obtain a completion queue handle, which is used for subsequent CQ references. An application then associates the CQ with the logical endpoints and with registered memory segments to be used for future data transactions. After initiating transactions between endpoints, an application references the associated CQ to track various events that are related to transaction completion, messaging notifications, and errors.

Completion queues have a fixed size which is specified when they are created. When the queue is full, it is said to be in the overrun state. CQEs received when the CQ is full are discarded.

Local completion queues track the completion of operations initiated on local endpoints. They are linked to these endpoints by being specified as a parameter to EpCreate(). See EpCreate on page 61.

Receive completion queues notify the application of completion of operations initiated on remote endpoints targeting local registered memory. They are linked with this memory by being specified as a parameter to MemRegister(). See MemRegister on page 52.

See Completion Queue Management on page 49.

### 2.1.3 Register Memory

Memory allocated by an application must be registered before it can be given to a peer as a destination buffer or used as a source buffer for most uGNI transactions.

Registration associates a specific memory segment (described by a pointer and a size) with an NIC that will be performing transactions from/to this memory. Memory can be registered with multiple NICs at the same time; it is up to the application to ensure that the NICs do not use the memory simultaneously.
When an application registers a memory segment, it receives a memory handle for subsequent references to that segment. The application attaches that segment to an NIC and must keep track of the handles for each attached NIC and deregister the associated memory when no longer needed. See Memory Registration on page 52.

### 2.1.4 Create Logical Endpoints

Before instances of an application can start communicating with each other, the logical local and remote endpoints have to be created. Endpoint properties include the handle of the NIC device used for this connection, the remote PE, and the local CQ. Applications usually synchronize among instances before attempting to bind endpoints. When no longer needed, the application must unbind the endpoint explicitly through a function call or implicitly by destroying the endpoint. See Logical Endpoint on page 61.

### 2.1.5 Transfer Data

There are two mechanisms for accessing remote memory on another node — *Fast Memory Access* (FMA), and *Block Transfer Engine* (BTE).

For some transfer operations, the GNI kernel driver first exchanges *datagrams*, which contain messaging parameters, to initialize communication between PEs.

#### 2.1.5.1 Fast Memory Access (FMA)

Use FMA primarily for the efficient transfer of small, possibly non-contiguous blocks of data between local and remote memory. For example, use FMA for the short inter-process data transfers typical of one-sided communication in models like Cray SHMEM, UPC or Coarray Fortran.

GNI implements FMA through a set of memory windows that enable data to be moved by the processor directly from user space, through the Gemini ASIC, to the network. Stores into an FMA window are used to generate remote memory reference requests. FMA functionality is divided into several categories as described below:

- **SMSG**—To send and receive point-to-point short messages (SMSG) between endpoints an application must first initialize an endpoint with communication parameters and preregistered buffers required for performing FMA transactions. An application then calls a send function with pointer, length and control information. An application calls a receive function to obtain a pointer to the header of the next available message for a given connection. Either the application process messages immediately or copies them to another buffer. The application must release the message buffer when it is no longer needed. See FMA Short Messaging (SMSG) on page 77.
Using the GNI and DMAPP APIs

MSGQ—To send and receive short messages for large jobs, use the Shared Message Queue (MSGQ) interface. For large jobs (exceeding 200,000 ranks), memory constraints limit the use of short messaging interface for all-to-all communication patterns. For increased scalability, the MSGQ interface allows all job instances on a node to share the message buffer resources required for an SMSG connection. Greater scalability is achieved because the number of point-to-point connections needed in a job will scale by the node count rather than by the number of ranks. MSGQ uses the SMSG facilities for sending and receiving messages, keeping SMSG control information and mailbox buffer space in a shared buffer for all job instances on a node to access. See Shared Message Queue (MSGQ) on page 85.

FMA DM—To access Distributed Memory (DM), moving user data between local and remote memory, an application prepares a Transaction Request Descriptor, which has properties such as type (PUT/GET), CQ, data source and destination, and length. To post the transaction, an application passes the pointer to a Transaction Request Descriptor to the PostFma function. See FMA DM on page 76.

AMO—To execute an atomic memory operation (AMO), an application prepares a Transaction Request Descriptor by specifying the remote node, the AMO command to execute, operands, and other fields, depending on the syntax of the AMO command. To post the transaction, the application passes the pointer to a Transaction Request Descriptor to the PostFma function. See FMA DM on page 76.

2.1.5.2 Block Transfer Engine (BTE)

The BTE functionality, which is implemented on the ASIC, is intended primarily for large asynchronous data transfers between nodes. More time is required to set up data for a transfer, than for an FMA transfer, but once initiated, there is no further involvement by the processor core.

An application can direct the Block Transfer Engine (BTE) to perform an RDMA PUT operation, which instructs the BTE to move data from local to a remote memory, or an RDMA GET operation, which instructs the BTE to move data from remote to local memory and to notify a source and destination upon completion. These functions write block transfer descriptors to a queue in the NIC, and the BTE services the requests asynchronously. Block transfer descriptors use privileged state, so access to the BTE is gated through the kernel. Due to the overhead of accessing the BTE through the operating system, the BTE mechanism is more appropriate for larger messages.

PUT/GET transactions require a pointer and a memory domain handle to identify the data source and destination, data length and return a transaction ID. These operations use several modes, some of which are appropriate for kernel-level applications because they bypass memory registration. Other modes are targeted for user applications which control data ordering, event notification, and synchronization. See RDMA (BTE) on page 94.


2.1.6 Process Completion Queue

The calling process must poll a completion queue for a completion entry to discover information about events, such as transaction completions, message notifications, and errors, which are generated by the NIC device. If a new completion entry is found, the application processes status information and event data.

Any error in the network that leads to data loss will result in the NIC's generating an interrupt and delivering an error to the completion queue associated with the logical endpoints of the transaction within that communication domain.

To avoid dropped completion notifications, applications should ensure that the number of operations posted on Endpoints attached to a src_cq_handle does not exceed the completion queue capacity at any time. See Completion Queue Processing on page 95.

2.2 Restrictions

The total number of GNI application processes running on a given node should be limited to the number of CPU cores of the node.

2.3 Compiling

To compile a routine that makes GNI calls, include the gni_pub.h header, which is contained in the gni-headers module. Other required headers are stdint.h and sys/types.h.

To link a routine that makes GNI calls, link with the ugni library, contained in the ugni module.
This chapter contains reference information for functions, structures, and enumerations contained in the GNI API. Your application must include the gni_pub.h file when using this API.

3.1 Naming Conventions

The GNI API defines four types of entities: functions, types, return codes and constants. User-level functions start with GNI_ and use mixed upper and lower case. Kernel-level functions start with gni_ and use lower case with underscores to separate words.

Only user-level GNI functions (uGNI) are documented in this guide.

3.2 Communication Domain

3.2.1 CdmCreate

The GNI_CdmCreate function creates an instance of the communication domain.

3.2.1.1 Synopsis

gni_return_t GNI_CdmCreate (  
    IN uint32_t inst_id,  
    IN uint8_t ptag,  
    IN uint32_t cookie,  
    IN uint32_t modes,  
    OUT gni_cdm_handle_t *cdm_handle)

3.2.1.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst_id</td>
<td>Rank of the instance in the job.</td>
</tr>
<tr>
<td>ptag</td>
<td>Protection tag.</td>
</tr>
<tr>
<td>cookie</td>
<td>Unique identifier generated by the system. Along with ptag, the cookie identifies the communication domain.</td>
</tr>
</tbody>
</table>
modes
The modes bit mask. The following flags are used for this parameter:

- One of the following mutually exclusive flags:
  - GNI_CDM_MODE_FORK_NOCOPY
  - GNI_CDM_MODE_FORK_PARTCOPY
  - GNI_CDM_MODE_FORK_FULLCOPY
- GNI_CDM_MODE_CACHED_FMA_SHARED
- GNI_CDM_MODE_CACHED_AMO_ENABLED
  Enable the use of cached AMO operations.
- GNI_CDM_MODE_CQ_NIC_LOCAL_PLACEMENT
  Allow a request for placement of the CQ in host memory closest to the NIC, which increases small message injection rate for some applications.
- GNI_CDM_MODE_DUAL_EVENTS
  Must be used when local and global completion events are needed for RDMA post operations.
- GNI_CDM_MODE_FAST_DATAGRAM_POLL
- GNI_CDM_MODE_BTE_SINGLE_CHANNEL
  Enable RDMA posts through one BTE channel, instead of defaulting to using all available channels.
- One of the following mutually exclusive flags:
  - GNI_CDM_MODE_ERR_NO_KILL
  - GNI_CDM_MODE_ERR_ALL_KILL

cdm_handle
Returns a pointer to a handle for the communication domain object. The handle is used by other functions to specify a particular instance of the communication domain.
3.2.1.3 Return Codes

GNI_RC_SUCCESS
The operation completed successfully.

GNI_RC_INVALID_PARAM
One of the input parameters was invalid.

GNI_RC_ERROR_NOMEM
Insufficient memory to complete the operation.

3.2.2 CdmDestroy
The GNI_CdmDestroy function destroys the instance of the communication domain and removes associations between the calling process and the Gemini NIC devices that were established by the corresponding GNI_CdmAttach function.

3.2.2.1 Synopsis

```c
gni_return_t GNI_CdmDestroy (
    IN gni_cdm_handle_t cdm_handle
);
```

3.2.2.2 Parameters

`cdm_handle` The communication domain handle.

3.2.2.3 Return Codes

GNI_RC_SUCCESS
The operation completed successfully.

GNI_RC_INVALID_PARAM
The caller specified an invalid communication domain handle.

3.2.3 CdmGetNicAddress
The CdmGetNicAddress function reads the /sys/class/gemini/ghalX/address file, where X is the device_id.

3.2.3.1 Synopsis

```c
gni_return_t GNI_CdmGetNicAddress ( 
    IN uint32_t device_id, 
    OUT uint32_t *address, 
    OUT uint32_t *cpu_id )
```
3.2.3.2 Parameters

**device_id**  
The device identifier. For example, the NIC /dev/kgni has the `device_id=DEVICE_MINOR_NUMBER-GEMINI_BASE_MINOR_NUMBER=1`.

**address**  
PE address of the NIC.

**cpu_id**  
ID of the first CPU in the slot directly connected to the NIC.

3.2.3.3 Return Codes

GNI_RC_SUCCESS  
The operation completed successfully.

GNI_RC_NO_MATCH  
The specified `device_id` does not exist.

3.2.4 CdmAttach

The `CdmAttach` function associates the communication domain with a Gemini NIC and provides a NIC handle to the upper layer protocol. A process cannot attach a single communication domain instance to a Gemini NIC more than once, but it can attach multiple communication domains to a single Gemini NIC.

If NTT is used, the `local_address` contains the virtual address of the PE, rather than its physical address.

3.2.4.1 Synopsis

```c
gni_return_t GNI_CdmAttach (  
  IN gni_cdm_handle_t cdm_handle,  
  IN uint32_t device_id,  
  OUT uint32_t *local_address,  
  OUT gni_nic_handle_t *nic_handle)
```

3.2.4.2 Parameters

**cdm_handle**  
Communication domain handle.

**device_id**  
Device identifier for the Gemini NIC to which the communication domain attaches. The device id is the minor number for the device that is assigned to the device by the system when the device is created. To determine the device number, look in the `/dev` directory, which contains a list of devices. For a NIC, the device is listed as `kgniX`, where `X` is the device number.
local_address

Returns a pointer to the PE address for the NIC that this function has attached to the communication domain. If NTT is used, the local_address contains the virtual address of the PE, rather than its physical address.

nic_handle

Returns a pointer to a handle for the NIC. The handle is used by the API to specify an instance of a Gemini NIC.

### 3.2.4.3 Return Codes

GNI_RC_SUCCESS

The operation completed successfully.

GNI_RC_INVALID_PARAM

The caller specified an invalid communication domain handle.

GNI_RC_NO_MATCH

The specified device_id does not exist.

GNI_RC_ERROR_RESOURCE

The operation failed due to insufficient resources. To resolve this, verify that the FMA descriptors are available on the given NIC. The most likely cause of this error is that too many CDM domains got attached to the given NIC on that node.

GNI_RC_ERROR_NOMEM

Insufficient memory to complete the operation.

GNI_RC_INVALID_STATE

The caller attempted to attach a communication domain instance to the Gemini NIC device more than once.

GNI_RC_PERMISSION_ERROR

Insufficient permissions to perform the operation.

### 3.2.5 GetVersion

The GetVersion function returns the version number of the uGNI library.

#### 3.2.5.1 Synopsis

```c
gni_return_t GNI_GetVersion(
    OUT uint32_t *version)
```
3.2.5.2 Parameters

*version*  
GNI version number.

3.2.5.3 Return Codes

GNI_RC_SUCCESS  
Operation completed successfully.

GNI_RC_INVALID_PARAM  
The version is undefined.

3.2.6 ConfigureNTT

The Node Translation Table (NTT) works in conjunction with the FMA mechanism to allow applications to employ logical network endpoints when addressing remote nodes. This facilitates efficient user-level access to FMA, as well as simplifying checkpoint/restart operations, etc. There are 8192 entries in the NTT for each NIC on the Gemini network ASIC. Each entry contains 18 bits of data which is used to convert an application virtual PE into a 16-bit Network Endpoint ID and a 2-bit Gemini core (DstID). Bit 17 of the entry specifies bit 1 of the DstID field. The NTTConfig register controls the granularity for NTT addressing.

The GNI_ConfigureNTT function sets up entries in the NTT associated with a particular /dev/kgni device.

If the table field of the input *ntt_desc* is set to NULL, the NTT entries starting from *ntt_base* up to and including *ntt_base+ntt_desc->group_size−1* are reset to 0.

If the *ntt_base* is −1 and *ntt_desc->group_size* is −1, and the table field of *ntt_desc* is NULL, all entries of NTT allocations not currently in use will be reset to 0.

3.2.6.1 Synopsis

```
gni_return_t GNI_ConfigureNTT (  
    IN uint32_t device_id,  
    IN gni_ntt_descriptor_t *ntt_desc,  
    OUT uint32_t ntt_base  
)
```

3.2.6.2 Parameters

*device_id*  
The device identifier, for example, for /dev/kgni1 it is *device_id* = DEVICE_MINOR_NUMBER − GEMINI_BASE_MINOR_NUMBER = 1

*ntt_desc*  
NTT configuration descriptor. Descriptions are set using the gni_ntt_descriptor structure which has the types found in Table 1.
### Table 1. gni_ntt_descriptor

<table>
<thead>
<tr>
<th>Type</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>group_size</td>
<td>Size of the NTT group to be configured.</td>
</tr>
<tr>
<td>uint8_t</td>
<td>granularity</td>
<td>NTT granularity.</td>
</tr>
<tr>
<td>uint32_t</td>
<td>table</td>
<td>Pointer to the array of new NTT values.</td>
</tr>
<tr>
<td>uint8_t</td>
<td>flags</td>
<td>Configuration flags.</td>
</tr>
</tbody>
</table>

**ntt_base**

On return, set to the base NTT entry allocated by the driver.

#### 3.2.6.3 Return Codes

**GNI_RC_SUCCESS**

The operation completed successfully.

**GNI_RC_INVALID_PARAM**

One of the input parameters was invalid.

**GNI_RC_PERMISSION_ERROR**

The process has insufficient permission to set up NTT resources.

**GNI_RC_ERROR_RESOURCE**

A hardware resource limitation prevents NTT setup.

**GNI_RC_ERROR_NOMEM**

Insufficient memory to complete the operation.

**GNI_RC_NO_MATCH**

The specified `device_id` does not exist.

#### 3.2.7 ConfigureJob

The `GNI_ConfigureJob` function sets the configuration options for the job, which include the device ID, the job ID, the protection tag, cookie, and limit values for the job. The user (ALPS) can call this function multiple times for the same Gemini interface. The driver looks up a triplet `(job_id+ptag+cookie)` and then adds a new entry into the list it maintains for each physical NIC, for every unique triplet. Each entry may have a non-unique `job_id` or `ptag` or `cookie`. Using the same `ptag` with a different `job_id` is illegal and such calls fail. This function must be called before `GNI_CdmAttach` for the CDM with the same `ptag+cookie`. Calling `GNI_ConfigureJob` for the same triplet has no effect, unless `limits` is non-NULL.
An application may also use this function to associate NTT resources with a job. The NTT resources would have been previously allocated by a call to GNI_ConfigureNTT. In this case, the application sets the \textit{ntt\_base} and \textit{ntt\_size} fields in the limits input. If the application expects the driver to clean up the NTT resources upon termination of the job, the application sets the \textit{ntt\_ctrl} field in the limits input to GNI\_JOB\_CTRL\_NTT\_CLEANUP. The application should not subsequently attempt to change \textit{ntt\_base} or \textit{ntt\_size} by calling \texttt{ConfigureJob} with different NTT parameters.

### 3.2.7.1 Synopsis

\begin{verbatim}
gni_return_t GNI_ConfigureJob (  
    IN uint32_t device_id,  
    IN uint64_t job_id,  
    IN uint8_t ptag,  
    IN uint32_t cookie,  
    IN gni_job_limits_t *limits )
\end{verbatim}

### 3.2.7.2 Parameters

\begin{itemize}
\item \textit{device\_id} \hspace{1cm} The device identifier, for example, for /dev/kgni1 has \textit{device\_id} = DEVICE\_MINOR\_NUMBER - GEMINI\_BASE\_MINOR\_NUMBER = 1.
\item \textit{job\_id} \hspace{1cm} Job container identifier.
\item \textit{ptag} \hspace{1cm} Protection tag to be used by all applications in the given job container.
\item \textit{cookie} \hspace{1cm} Unique identifier. Assigned to all applications within the job container along with \textit{ptag}.
\item \textit{limits} \hspace{1cm} When this argument is non-NULL, the driver takes all the limit values that are not set to GNI\_JOB\_INVALID\_LIMIT and stores them into the table indexed by the \textit{ptag}. These limits are imposed on all applications running within the given job container. If you set different limits for the same \textit{ptag}, the driver overwrites previously set limits.
\end{itemize}
3.2.7.3 Return Codes

GNI_RC_SUCCESS
The operation completed successfully.

GNI_RC_INVALID_PARAM
One of the input parameters was invalid.

GNI_RC_PERMISSION_ERROR
The process has insufficient permission to configure job or no NTT entries exist for input \textit{ntt\_base/ntt\_size} fields in the limits argument.

GNI_RC_NO_MATCH
The specified \textit{device\_id} does not exist or there are no NTT entries.

GNI_RC_INVALID_STATE
The caller attempted to use the same \textit{ptag} with a different \textit{job\_id} or a different \textit{cookie}.

GNI_RC_ILLEGAL_OP
The application is attempting to resize the NTT resources.

GNI_RC_ERRORRESOURCE
A resource allocation error was encountered while trying to configure the job resources.

GNI_RC_ERROR_NOMEM
Insufficient memory to complete the operation.

3.2.8 ConfigureNTTandJob

This function combines the GNI\_ConfigureNTT and GNI\_ConfigureJob functions; it sets up entries in the NTT associated with a particular /dev/kgni device, and then sets up configuration options of the job in a single system call. Setting ntt\_desc to NULL will makes this function equivalent to GNI\_ConfigureJob.

This function cannot be used to clear the NTT table; GNI\_ConfigureNTT should be used.

This function or GNI\_ConfigureJob must be called before GNI\_CdmAttach for the CDM with the same \textit{ptag+cookie}.

Calling GNI\_ConfigureNTTandJob for the same triplet has no effect, unless \textit{limits} is non-NULL.
If the application expects the driver to clean up the NTT resources upon termination of the job, the application sets the `ntt_ctrl` field in the limits input to `GNI_JOB_CTRL_NTT_CLEANUP`. The application should not attempt to change `ntt_base` or `ntt_size` by calling `ConfigureNTTandJob` subsequently with different NTT parameters.

### 3.2.8.1 Synopsis

```c
gni_return_t GNI_ConfigureNTTandJob (
    IN uint32_t device_id,
    IN uint64_t job_id,
    IN uint8_t ptag,
    IN uint32_t cookie,
    IN gni_job_limits_t *limits,
    IN gni_ntt_descriptor_t *ntt_desc,
    OUT uint32_t ntt_base )
```

### 3.2.8.2 Parameters

- **device_id**: The device identifier, for example, for `/dev/kgni1` has `device_id = DEVICE_MINOR_NUMBER - GEMINI_BASE_MINOR_NUMBER = 1`.
- **job_id**: Job container identifier.
- **ptag**: Protection tag to be used by all applications in the given job container.
- **cookie**: Unique identifier. Assigned to all applications within the job container along with `ptag`.
- **limits**: The driver takes all the limit values that are not set to `GNI_JOB_INVALID_LIMIT` and stores them into the table indexed by the `ptag`. These limits are imposed on all applications running within the given job container. If you set different limits for the same `ptag`, the driver overwrites previously set limits.
- **ntt_desc**: NTT configuration descriptor. Descriptions are set using the `gni_ntt_descriptor` structure which has the types found in Table 1.
- **ntt_base**: On return, set to the base NTT entry allocated by the driver.
3.2.8.3 Return Codes

GNI_RC_SUCCESS

The operation completed successfully.

GNI_RC_INVALID_PARAM

One of the input parameters was invalid.

GNI_RC_PERMISSION_ERROR

The process has insufficient permission to configure job or no NTT entries exist for input ntt_base/ntt_size fields in the limits argument.

GNI_RC_NO_MATCH

The specified device_id does not exist or there are no NTT entries.

GNI_RC_INVALID_STATE

The caller attempted to use the same ptag with a different job_id or a different cookie.

GNI_RC_ILLEGAL_OP

The application is attempting to resize the NTT resources.

GNI_RC_ERROR_RESOURCE

A resource allocation error was encountered while trying to configure the job resources.

GNI_RC_ERROR_NOMEM

Insufficient memory to complete the operation.

3.3 Completion Queue Management

3.3.1 CqCreate

The CqCreate function creates a new completion queue. The caller must specify the minimum number of completion entries that the queue must contain in the entry_count parameter. To avoid dropped completion notifications, you should set up your application to verify that the number of operations posted on endpoints attached to a cq_handle does not exceed the completion queue capacity at any time.

The event_handler function, if specified, is called if (and only if) CqGetEvent or CqWaitEvent return with either GNI_RC_SUCCESS or GNI_RC_TRANSACTION_ERROR. The handler is invoked at some time between the time that the CQ entry arrives in the CQ, and the successful return of GNI_CqGetEvent or GNI_CqWaitEvent.
The user must call GNI_CqGetEvent or GNI_CqWaitEvent for each event deposited into the CQ, regardless of whether an event_handler is used.

Completion queues may be used for the receipt of locally generated events, such as those arising from GNI_Post style transactions or may be used for the receipt of remote events, but not both.

### 3.3.1.1 Synopsis

```c
gni_return_t GNI_CqCreate (
    IN gni_nic_handle_t nic_handle,
    IN uint32_t entry_count,
    IN uint32_t delay_count,
    IN uint32_t mode,
    IN void (*event_handler)(gni_cq_entry_t *, void *),
    IN void *context,
    OUT gni_cq_handle_t *cq_handle)
```

### 3.3.1.2 Parameters

- **nic_handle** The handle of the associated Gemini NIC.
- **entry_count** The number of completion entries that this completion queue holds.
- **delay_count** The number of events the NIC allows before generating an interrupt. Setting this parameter to zero results in interrupt delivery with every event. When using this parameter, the **mode** parameter must be set to GNI_CQ_BLOCKING.
- **mode** The mode of operation for the new completion queue. The following modes are used by this parameter:
  - GNI_CQ_BLOCKING
  - GNI_CQ_NOBLOCK
- **event_handler** User-supplied callback function to be run for each CQ entry received in the CQ. The handler is supplied with two arguments: a pointer to the CQ entry, and a pointer to the context provided at CQ creation.
- **context** User-supplied pointer that is passed to the handler callback function.
- **cq_handle** Returns a pointer to the handle of the newly created completion queue.
3.3.1.3 Return Codes

GNI_RC_SUCCESS
A new completion queue was successfully created.

GNI_RC_INVALID_PARAM
One or more of the parameters was invalid.

GNI_RC_ERROR_RESOURCE
The completion queue could not be created due to insufficient resources.

GNI_RC_ERROR_NOMEM
Insufficient memory to complete the operation.

3.3.2 CqDestroy

The CqDestroy function destroys a specified completion queue. If any endpoints are associated with the completion queue, the completion queue is not destroyed and an error is returned.

3.3.2.1 Synopsis

gni_return_t GNI_CqDestroy (  
    IN gni_cq_handle_t cq_handle)

3.3.2.2 Parameters

cq_handle The handle for the completion queue to be destroyed.

3.3.2.3 Return Codes

GNI_RC_SUCCESS
The completion queue was successfully destroyed.

GNI_RC_INVALID_PARAM
The cq_handle was invalid.

GNI_RC_ERROR_RESOURCE
The completion queue could not be destroyed because one or more endpoint instances are still associated with it. Use EpDestroy to destroy the endpoint instance, then try calling this function again.
3.4 Memory Registration

Only registered memory is remotely accessible. After an application allocates a memory region to be used as a source or destination buffer for data transfers, it must register memory with the NIC to support the remote address translation and data protection mechanism.

3.4.1 MemRegister

The `MemRegister` function allows a process to register a region of memory with the NIC. It requires the specification of a handle for the currently open NIC, the starting address of the memory region to be registered, its length, a flag identifying the region as read or read-write. A completion queue handle may be optionally specified. A new memory handle is generated for each region of memory that is registered by a process. The content of the memory region being registered is not altered.

Memory registration typically involves allocating a Memory Domain Descriptor (MDD). Each MDD provides the base address and bounds of a local memory region which is also referred to as a local memory window. Depending on the allocated memory region, the function then creates a sufficient number of associated Memory Relocation Table (MRT) entries on the NIC or GART entries on the AMD processor to span the allocated memory region. The user may specify an arbitrary size region of memory, with arbitrary alignment, but the actual area of memory registered will be on MRT block granularity when using the MRT (or physical page granularity if MRT is not enabled for this process).

Incoming network packets referencing memory include a memory domain handle (MDH), which indexes into the table of memory domain descriptors (MDDs). The address contained within the network packet is added to the base, and checked against the limit. In other words, the address is used as an offset into a local memory window defined by the MDD. This allows the local node to place the memory associated with a given MDD in any location in its local memory space using the associated MRT or GART entry.

Users should ordinarily choose a single-segment memory registration to register application memory, and reserve multiple-segment registration for special cases. Using a single segment to register a memory region allows an application to use a virtual address in future transactions in and out of the registered region. Using multiple segments during the registration requires the application to use an offset within the registered memory region instead of a virtual address in all future transactions, where the registered region is aligned to MRT block size (or page size for non-MRT registrations).
MDHs are partially specified by the user level software and cannot be trusted by the NIC driver, so each MDD also contains a protection tag (PTag) which is assigned by the operating system and cannot be modified by the user. The PTag in an incoming memory reference is checked against a PTag stored in the referenced MDD, to verify that the reference is permitted to use that MDD.

3.4.1.1 Virtual Memory Domain Handles

The memory registration mechanism also supports a capability to use virtual Memory Domain Handles (vMDH), which supports Distributed Memory (DM) programming models.

For this discussion, a DM programming model is defined as a parallel job consisting of multiple independent processes distributed across one or more nodes of a Cray XE system. The processes may be executing the same application or different ones. At least one memory segment of equal size on each node is made remotely accessible by all of the processes in the job. To implement this model, the Virtual Memory Domain Handle Table (vMDHT) creates a relationship between the virtual MDH in the incoming network request and the actual memory domain handle to use in looking up the MDD associated with the incoming reference.

3.4.1.2 Synopsis

gni_return_t GNI_MemRegister (  
  IN gni_nic_handle_t nic_handle,  
  IN uint64_t address,  
  IN uint64_t length,  
  IN gni_cq_handle_t dst_cq_handle,  
  IN uint32_t flags,  
  IN uint32_t vmdh_index,  
  INOUT gni_mem_handle_t *mem_handle)
3.4.1.3 Parameters

nic_handle    Handle of a currently open Gemini NIC.

address       Starting address of the memory region to be registered.

length        Length of the memory region to be registered, in bytes. A length parameter of zero will result in a GNI_RC_INVALID_PARAM error.

dst_cq_handle If this value is not NULL, it specifies the completion queue to receive events related to the transactions initiated by the remote node into this memory region.

flags         Attributes of the memory region. A combination of the following flags are used for this parameter:

GNI_MEM_READWRITE

The read/write attribute is associated with the memory region.

GNI_MEM_READ_ONLY

The read only attribute is associated with the memory region.

GNI_MEM_USE_VMDH

Directive to use virtual MDH while registering this memory region. If the GNI_MEM_USE_VMDH flag is set, this function will fail if GNI_SetMddResources has not been called to specify the size of the MDD block to be used. If the GNI_MEM_USE_VMDH flag is set, this function will fail with GNI_RC_ERROR_RESOURCE return code if the vMDH entry specified by vmdh_index is already in use.

GNI_MEM_USE_GART

Directive to use GART while registering the memory region.
**GNI_MEM_RELAXED_PI_ORDERING**

Instructs the NIC to allow relaxed HT ordering for Non-Posted and Posted write requests into the processor by enabling rules for both *Non-posted pass posted writes* and *Posted pass posted writes*. Non-posted pass posted writes are enabled by default. Use this attribute with caution as it may affect transaction ordering. It is overridden by **GNI_MEM_STRICT_PI_ORDERING**.

**GNI_MEM_STRICT_PI_ORDERING**

Instructs the NIC to enforce strict HT ordering for the memory region.

**GNI_MEM_PI_FLUSH**

Instructs the NIC to issue a HT FLUSH command prior to sending network responses for the memory region.

**vmdh_index**

Specifies the index within the preallocated memory domain descriptor block that must be used for the registration. For example, when this parameter is set to 0, it uses the first entry of the memory domain descriptor block. If set to −1, it relies on the GNI library to allocate the next available entry from the memory domain descriptor block.

**mem_handle**

The new memory handle for the region.

### 3.4.1.4 Return Codes

**GNI_RC_SUCCESS**

The memory region was successfully registered.

**GNI_RC_INVALID_PARAM**

One of the input parameters was invalid.

**GNI_RC_ERROR_RESOURCE**

The registration operation failed due to insufficient resources.

**GNI_RC_ERROR_NOMEM**

Insufficient memory to complete the operation.

**GNI_RC_PERMISSION_ERROR**

The user's buffer read/write permissions conflict with the flags argument.
3.4.2 MemRegisterSegments

The MemRegisterSegments function enables a process to register a memory region that is composed of multiple memory segments with the NIC.

Multiple segment registration should be reserved for special cases. Single segment memory registration is the preferred method for memory registration. To register a single segment, use GNI_MemRegister.

The user may specify an arbitrary size region of memory, with arbitrary alignment, but the actual area of memory registered will be registered on MRT block granularity (or physical page granularity if MRT is not enabled for this process).

If an application registers multiple segments, it must use an offset within the registered memory region instead of a virtual address in all future transactions where registered region is aligned to MRT block size (or page size for non-MRT registrations). This is because a single memory domain is used for the registration of multiple segments and transactions must access memory for these segments as if it was contiguous.

A new memory handle is generated for each region of memory that is registered by a process.

3.4.2.1 Synopsis

```c
gni_return_t GNI_MemRegisterSegments (    
    IN gni_nic_handle_t nic_handle,    
    IN gni_mem_segment_t *mem_segments,    
    IN uint32_t segments_cnt,    
    IN gni_cq_handle_t dst_cq_handle,    
    IN uint32_t flags,    
    IN uint32_t vmdh_index,    
    INOUT gni_mem_handle_t *mem_handle)
```
3.4.2.2 Parameters

**nic_handle**
Handle of a currently open Gemini NIC.

**mem_segments**
List of segments to register. Each element of the list consists of the starting address of the memory region and the length, in bytes. The list elements are set using the `gni_mem_segment` structure.

**segment_cnt**
Number of segments in the `mem_segments` list.

**dst_cq_handle**
If this value is not `NULL`, it specifies the completion queue to receive events related to the transactions initiated by the remote node into this memory region.

**flags**
Attributes of the memory region. A combination of the following flags are used for this parameter:

- **GNI_MEM_READWRITE**
  The read/write attribute is associated with the memory region.

- **GNI_MEM_READ_ONLY**
  The read only attribute is associated with the memory region.

- **GNI_MEM_USE_VMDH**
  Directive to use virtual MDH while registering this memory region. If set, and `GNI_SetMddResources` has not been called to specify the size of the MDD block to be used, or if the vMDH entry specified by `vmdh_index` is already in use, this function will return the `GNI_RC_ERROR_RESOURCE` error code.

- **GNI_MEM_USE_GART**
  Directive to use GART while registering the memory region.
Using the GNI and DMAPP APIs

GNI_MEM_RELAXED_PI_ORDERING

Instructs the NIC to allow relaxed HT ordering for non-posted and posted write requests into the processor by enabling rules for both Non-posted pass posted writes and Posted pass posted writes. Non-posted pass posted writes are enabled by default.

GNI_MEM_STRICT_PI_ORDERING

Instructs the NIC to enforce strict HT ordering for the memory region.

GNI_MEM_PI_FLUSH

Instructs the NIC to issue a HT FLUSH command prior to sending network responses for the memory region.

vmdh_index

Specifies the index within the preallocated memory domain descriptor block that must be used for the registration. For example, when this parameter is set to 0, it uses the first entry of the memory domain descriptor block. If set to -1, it relies on the GNI library to allocate the next available entry from the memory domain descriptor block.

mem_handle

The new memory handle for the region.

3.4.2.3 Return Codes

GNI_RC_SUCCESS

The memory region was successfully registered.

GNI_RC_INVALID_PARAM

One on the parameters was invalid.

GNI_RC_ERROR_RESOURCE

The registration operation failed due to insufficient resources.

GNI_RC_ERROR_NOMEM

Insufficient memory to complete the operation.

GNI_RC_PERMISSION_ERROR

The user’s buffer read/write permissions conflict with the flags argument.
3.4.3 SetMddResources

The SetMddResources function specifies the size of a contiguous block of MDD entries that can be used for future memory registrations.

3.4.3.1 Synopsis

```c
gni_return_t GNI_SetMddResources (  
   IN gni_nic_handle_t nic_handle,  
   IN uint32_t num_entries
);
```

3.4.3.2 Parameters

- `nic_handle` The handle for the NIC.
- `num_entries` Number of MDD entries in the block.

3.4.3.3 Return Codes

- `GNI_RC_SUCCESS` The block size was successfully specified.
- `GNI_RC_INVALID_PARAM` One or more of the parameters was invalid.
- `GNI_RC_ERROR_NOMEM` Insufficient memory to complete the operation.

3.4.4 MemDeregister

The MemDeregister function deregisters memory that was previously registered and unlocks the associated pages from physical memory. An application may decide to deregister memory upon transaction completion or keep memory for future transactions. The contents and attributes of the deregistered region of memory are not altered in any way.

3.4.4.1 Synopsis

```c
gni_return_t GNI_MemDeregister (  
   IN gni_nic_handle_t nic_handle,  
   IN gni_mem_handle_t *mem_handle
);
```
3.4.4.2 Parameters

- `nic_handle`: The handle for the NIC that owns the memory region being deregistered.
- `mem_handle`: Memory handle for the region; obtained from a previous call to MemRegister.

3.4.4.3 Return Codes

- `GNI_RC_SUCCESS`: The memory region was successfully deregistered.
- `GNI_RC_INVALID_PARAM`: One or more of the parameters was invalid.

3.4.5 MemHndlQueryAttr

Query for memory handle attributes while testing the memory handle for correctness. Only one attribute may be tested at a time.

3.4.5.1 Synopsis

```c
gni_return_t GNI_MemHndlQueryAttr (  
    IN gni_mem_handle_t *mem_handle,  
    IN gni_mem_handle_attr_t attr  
    OUT int *yesno);  
```

3.4.5.2 Parameters

- `mem_handle`: Memory handle for the region, obtained from a previous call to MemRegister.
- `attr`: Attribute of `mem_handle` to test for. See `gni_mem_handle_attr` on page 112.
- `yesno`: Returns 1 if `mem_handle` is described by the attribute, otherwise 0.

3.4.5.3 Return Codes

- `GNI_RC_SUCCESS`: The memory region was successfully queried.
- `GNI_RC_INVALID_PARAM`: One or more of the parameters was invalid.
3.5 Logical Endpoint

3.5.1 EpCreate

The EpCreate function creates an instance of a logical endpoint. A new instance is always created in a non-bound state. A non-bound endpoint is able to exchange posted data with any bound remote endpoint within the same communication domain. An endpoint cannot be used to post RDMA or FMA transactions or to send short messages while it is in a non-bound state.

3.5.1.1 Synopsis

gni_return_t GNI_EpCreate (  
    IN gni_nic_handle_t nic_handle,  
    IN gni_cq_handle_t src_cq_handle,  
    OUT gni_ep_handle_t *ep_handle)

3.5.1.2 Parameters

nic_handle
    Handle of the associated Gemini NIC.

src_cq_handle
    Handle of the completion queue that is used by default to deliver events related to the transactions initiated by the local node.

ep_handle
    Returns a pointer to the handle of the newly created endpoint instance.

3.5.1.3 Return Codes

GNI_RC_SUCCESS
    Operation completed successfully.

GNI_RC_INVALID_PARAM
    One of the input parameters was invalid.

GNI_RC_ERROR_NOMEM
    Insufficient memory to complete the operation.

3.5.2 EpSetEventData

The EpSetEventData function enables the user to define the values that the EpBind function uses to generate CQ events. By default, EpBind uses the Communication Domain's inst_id as the event data for generating global and remote CQ events and the Endpoint's remote_id for generating local CQ events.
3.5.2.1 Synopsis

```c
gni_return_t GNI_EpSetEventData (  
    IN gni_ep_handle_t ep_handle,  
    IN uint32_t local_event,  
    IN uint32_t remote_event)
```

3.5.2.2 Parameters

- **ep_handle**: The handle of the endpoint instance to define event data.
- **local_event**: The value to use when generating local CQ events.
- **remote_event**: The value to use when generating global and remote CQ events.

3.5.2.3 Return Codes

- **GNI_RC_SUCCESS**: Operation completed successfully.
- **GNI_RC_INVALID_PARAM**: An invalid endpoint handle was specified.

3.5.3 EpBind

The `EpBind` function binds a logical endpoint to a specific remote address and remote instance within the communication domain. Once bound, the endpoint can be used to post RDMA and FMA transactions.

3.5.3.1 Synopsis

```c
gni_return_t GNI_EpBind (  
    IN gni_ep_handle_t ep_handle,  
    IN uint32_t remote_addr,  
    OUT uint32_t remote_id)
```
3.5.3.2 Parameters

- **ep_handle**: Handle of the endpoint instance to be bound.
- **remote_addr**: Physical address of the Gemini NIC at the remote peer or NTT index, when NTT is enabled for the given communication domain.
- **remote_id**: User-specified ID of the remote instance in the job or the unique identifier of the remote instance within the upper layer protocol domain.

3.5.3.3 Return Codes

- **GNI_RC_SUCCESS**: Operation completed successfully.
- **GNI_RC_INVALID_PARAM**: One of the input parameters was invalid.
- **GNI_RC_ERROR_RESOURCE**: Failed due to insufficient resources.
- **GNI_RC_ERROR_NOMEM**: Insufficient memory to complete the operation.

3.5.4 EpUnbind

The EpUnbind function unbinds a logical endpoint from the specific address and remote instance, and releases any internal short message resources. A non-bound endpoint can exchange posted data with any bound remote endpoint within the same communication domain. An endpoint cannot be used to post RDMA, FMA transactions, or send short messages while it is in a non-bound state.

3.5.4.1 Synopsis

```c
gni_return_t GNI_EpUnbind (
    IN gni_ep_handle_t ep_handle)
```

3.5.4.2 Parameters

- **ep_handle**: The handle of the endpoint instance to be unbound.
3.5.4.3 Return Codes

GNI_RC_SUCCESS
Operation completed successfully.

GNI_RC_INVALID_PARAM
An invalid endpoint handle was specified.

GNI_RC_NOT_DONE
The endpoint still has outstanding transaction requests or pending datagrams and cannot be unbound at this time. Retry unbinding later.

3.5.5 EpDestroy

The `EpDestroy` function destroys an endpoint, cancels any outstanding requests, and releases short messaging resources.

3.5.5.1 Synopsis

```c
gni_return_t GNI_EpDestroy (
    IN gni_ep_handle_t ep_handle)
```

3.5.5.2 Parameters

`ep_handle` The handle of the endpoint instance to be destroyed.

3.5.5.3 Return Codes

GNI_RC_SUCCESS
Operation completed successfully.

GNI_RC_INVALID_PARAM
An invalid endpoint handle was specified.

3.5.6 EpPostData

The `EpPostData` function posts a datagram to be exchanged with a remote, bound endpoint in the communication domain.

3.5.6.1 Synopsis

```c
gni_return_t GNI_EpPostData (  
    IN gni_ep_handle_t ep_handle,  
    IN void *in_data,  
    IN uint16_t data_len,  
    IN void *out_buf,  
    IN uint16_t buf_size)
```
3.5.6.2 Parameters

```
ep_handle    Handle of the local endpoint.
in_data      Pointer to the data to send.
data_len      Size of the data to send, in bytes.
out_buf       Pointer to the buffer that receives the incoming datagram.
buf_size      Size of the buffer for the incoming datagram, in bytes.
```

3.5.6.3 Return Codes

```
GNI_RC_SUCCESS
The posted datagram is queued.

GNI_RC_INVALID_PARAM
The specified endpoint handle is invalid.

GNI_RC_ERROR_RESOURCE
The system allows only a single outstanding datagram transaction for each endpoint. There is already a pending datagram for the specified endpoint

GNI_RC_ERROR_NOMEM
Insufficient memory to complete transaction.

GNI_RC_SIZE_ERROR
The size of the datagram is too large.
```

3.5.7 EpPostDataWId

The EpPostDataWId function posts a datagram with a user-specified datagram_id to be exchanged with a remote endpoint in the communication domain. If the local endpoint is unbound, a datagram can be exchanged with any bound endpoint within the communication domain.

It is required that datagrams posted on unbound endpoints be associated with a datagram_id.
3.5.7.1 Synopsis

```c
gni_return_t GNI_EpPostDataWId (
    IN gni_ep_handle_t ep_handle,
    IN void *in_data,
    IN uint16_t data_len,
    IN void *out_buf,
    IN uint16_t buf_size,
    IN uint64_t datagram_id)
```

3.5.7.2 Parameters

- `ep_handle` Handle of the local endpoint.
- `in_data` Pointer to the data to send.
- `data_len` Size of the data to send.
- `out_buf` Pointer to the buffer that receives the incoming datagram.
- `buf_size` Size of the buffer for the incoming datagram.
- `datagram_id` Id associated with the datagram.

3.5.7.3 Return Codes

- **GNI_RC_SUCCESS**
  The posted datagram is queued.
- **GNI_RC_INVALID_PARAM**
  The specified endpoint handle is invalid, or an invalid value for the `datagram_id` was specified.
- **GNI_RC_ERROR_RESOURCE**
  The system allows only a single outstanding datagram transaction for each endpoint.
- **GNI_RC_ERROR_NOMEM**
  Insufficient memory to complete transaction.
- **GNI_RC_SIZE_ERROR**
  The size of the datagram is too large.

3.5.8 EpPostDataTest

The `EpPostDataTest` function returns the state of the `EpPostData` transaction.
3.5.8.1 Synopsis

```c
gni_return_t GNI_EpPostDataTest (  
    IN gni_ep_handle_t ep_handle,  
    OUT gni_ep_post_state_t *post_state,  
    OUT uint32_t *remote_address,  
    OUT uint32_t *remote_id)
```

3.5.8.2 Parameters

- **ep_handle**
  Handle of the local endpoint.

- **post_state**
  Returns a pointer to the state of the transaction. The following states are used for this parameter:
  - GNI_POST_PENDING
  - GNI_POST_COMPLETED
  - GNI_POST_ERROR
  - GNI_POST_TIMEOUT
  - GNI_POST_TERMINATED
  - GNI_POST_REMOTE_DATA

- **remote_address**
  Returns a pointer to the physical address of the Gemini NIC being used by the remote peer. The address is valid only if the `post_state` returns GNI_POST_COMPLETE.

- **remote_id**
  Returns a pointer to the user specific ID of the remote instance in the job. The ID is valid only if the `post_state` returns GNI_POST_COMPLETE.
3.5.8.3 Return Codes

GNI_RC_SUCCESS
The operation completed successfully.

GNI_RC_INVALID_PARAM
An invalid endpoint handle was specified.

GNI_RC_NO_MATCH
No matching datagram was found.

GNI_RC_SIZE_ERROR
The size of the output buffer is too small for the received datagram.

GNI_RC_ERROR_NOMEM
Insufficient memory to complete the operation.

3.5.9 EpPostDataTestById

The EpPostDataTestById function returns the state of the EpPostData transaction with the specified datagram_id.

3.5.9.1 Synopsis

gni_return_t GNI_EpPostDataTestById (
    IN gni_ep_handle_t ep_handle,
    IN uint64_t datagram_id,
    OUT gni_ep_post_state_t *post_state,
    OUT uint32_t *remote_address,
    OUT uint32_t *remote_id)

3.5.9.2 Parameters

ep_handle           Handle of the local endpoint. Must be the same as that used when posting the datagram using EpPostDataWId.

datagram_id          Id of the datagram to test for.

post_state           Returns a pointer to the state of the transaction. The following states are used for this parameter:
                      • GNI_POST_PENDING
                      • GNI_POST_COMPLETED
                      • GNI_POST_ERROR
                      • GNI_POST_TIMEOUT
remote_address

Returns a pointer to the physical address of the Gemini NIC being used by the remote peer. The address is only valid if the post_state returns GNI_POST_COMPLETE.

remote_id

Returns a pointer to the user specific ID of the remote instance in the job. The ID is only valid if the post_state returns GNI_POST_COMPLETE.

3.5.9.3 Return Codes

GNI_RC_SUCCESS

The operation completed successfully.

GNI_RC_INVALID_PARAM

An invalid endpoint handle was specified.

GNI_RC_NO_MATCH

No matching datagram was found.

GNI_RC_SIZE_ERROR

The size of the output buffer is too small for the received datagram.

GNI_RC_ERROR_NOMEM

Insufficient memory to complete the operation.

3.5.10 EpPostDataWait

The EpPostDataWait function is used to determine the result of a previously posted EpPostData call on the specified endpoint, blocking the calling thread until the completion of the posted transaction or until the specified timeout expires.

3.5.10.1 Synopsis

gni_return_t GNI_EpPostDataWait(
    IN gni_ep_handle_t ep_handle,
    IN uint32_t timeout,
    OUT gni_post_state_t *post_state,
    OUT uint32_t *remote_address,
    OUT uint32_t *remote_id)
3.5.10.2 Parameters

- **ep_handle**: Handle of the local endpoint.
- **timeout**: The length of time (in milliseconds) that this function waits for the transaction to complete. If a timeout is not needed, set this parameter to -1. If the timeout is set to zero, the system returns the GNI_RC_INVALID_PARAM error.
- **post_state**: State of the transaction
- **remote_address**: Physical address of the Gemini NIC at the remote peer. Valid only if post_state returned GNI_POST_COMPLETE.
- **remote_id**: User specific ID of the remote instance in the job (user). Unique address of the remote instance within the upper layer protocol domain (kernel). Valid only if post_state returned GNI_POST_COMPLETE.

3.5.10.3 Return Codes

- **GNI_RC_NO_MATCH**: No matching datagram was found.
- **GNI_RC_SUCCESS**: The transaction completed successfully.
- **GNI_RC_INVALID_PARAM**: The specified endpoint handle is invalid or timeout was set to zero.
- **GNI_RC_TIMEOUT**: The timeout expired before a successful completion of the transaction.
- **GNI_RC_SIZE_ERROR**: Output buffer is too small for the size of the received datagram.
- **GNI_RC_ERROR_NOMEM**: Insufficient memory to complete the operation.
3.5.11 EpPostDataWaitById

The EpPostDataWaitById function determines the result of a previously posted EpPostData call on the specified endpoint, blocking the calling thread until the transaction involving datagram_id has completed, or until the specified timeout expires.

3.5.11.1 Synopsis

```c
gni_return_t GNI_EpPostDataWaitById (
    IN gni_ep_handle_t ep_handle,
    IN uint64_t datagram_id,
    IN uint32_t timeout,
    OUT gni_ep_post_state_t *post_state,
    OUT uint32_t *remote_address,
    OUT uint32_t *remote_id)
```

3.5.11.2 Parameters

- `ep_handle` Handle of the local endpoint.
- `datagram_id` Id of the datagram to wait for.
- `timeout` The length of time (in milliseconds) that this function waits for the transaction to complete. If a timeout is not needed, set this parameter to -1. If the timeout is set to zero, the system returns the GNI_RC_INVALID_PARAM error.
- `post_state` Returns a pointer to the state of the transaction. The following states are used for this parameter:
  - GNI_POST_PENDING
  - GNI_POST_COMPLETED
  - GNI_POST_ERROR
  - GNI_POST_TIMEOUT
  - GNI_POST_TERMINATED
  - GNI_POST_REMOTE_DATA
- `remote_address` Returns a pointer to the physical address of the Gemini NIC being used by the remote peer. The address is only valid if the `post_state` returns GNI_POST_COMPLETE.
- `remote_id` Returns a pointer to the user specific ID of the remote instance in the job. The ID is only valid if the `post_state` returns GNI_POST_COMPLETE.
### 3.5.11.3 Return Codes

- **GNI_RC_SUCCESS**
  
  The transaction completed successfully.

- **GNI_RC_INVALID_PARAM**
  
  An invalid endpoint handle was specified or timeout was set to zero, or invalid datagram id was specified.

- **GNI_RC_TIMEOUT**
  
  The timeout expired before a successful completion of the transaction.

- **GNI_RC_SIZE_ERROR**
  
  The size of the output buffer is too small for the received datagram.

- **GNI_RC_NO_MATCH**
  
  No matching datagram found.

- **GNI_RC_ERROR_NOMEM**
  
  Insufficient memory to complete the operation.

### 3.5.12 EpPostDataCancel

The `EpPostDataCancel` function cancels an outstanding post data transaction.

#### 3.5.12.1 Synopsis

```c
gni_return_t GNI_EpPostDataCancel (
    IN gni_ep_handle_t ep_handle
)
```

#### 3.5.12.2 Parameters

- **ep_handle**  
  Handle of the local endpoint.

#### 3.5.12.3 Return Codes

- **GNI_RC_SUCCESS**
  
  The transaction cancellation was successful.

- **GNI_RC_INVALID_PARAM**
  
  The `ep_handle` parameter is invalid.

- **GNI_RC_NO_MATCH**
  
  No active post data transaction on the `ep_handle`. 
3.5.13 EpPostDataCancelById

The EpPostDataCancelById function cancels an outstanding post data transaction with the specified datagram Id.

3.5.13.1 Synopsis

gni_return_t GNI_EpPostDataCancelById(
    IN gni_ep_handle_t ep_handle,
    IN uint64_t datagram_id)

3.5.13.2 Parameters

ep_handle Handle of the local endpoint.

datagram_id Id of the datagram to cancel.

3.5.13.3 Return Codes

GNI_RC_SUCCESS
    The transaction cancellation was successful.

GNI_RC_INVALID_PARAM
    One of the input parameters are invalid.

GNI_RC_NO_MATCH
    No active post data transaction with the specified Id on the ep_handle.

3.5.14 PostDataProbe

The PostDataProbe function returns the remote ID and remote address of the first datagram found in completed, timed out, or canceled state for the CDM associated with the input NIC handle. This function must be used in conjunction with GNI_EpPostDataTestById or GNI_EpPostDataWaitById to obtain data exchanged in the datagram transaction.

3.5.14.1 Synopsis

gni_return_t GNI_PostDataProbe(
    IN gni_nic_handle_t nic_handle,
    OUT uint32_t *remote_address,
    OUT uint32_t *remote_id)
3.5.14.2 Parameters

- **nic_handle**: Handle of the NIC associated with the CDM for which the datagram status is being probed.

- **remote_address**: Physical address of the Gemini NIC at the remote peer with a datagram in the completed, timed-out or cancelled state. Valid only if the return value is GNI_RC_SUCCESS.

- **remote_id**: User specific ID of the remote instance in the upper layer protocol domain with a datagram in the completed, timed-out or cancelled state. Valid only if the return value is GNI_RC_SUCCESS.

3.5.14.3 Return Codes

- **GNI_RC_SUCCESS**: A datagram in the completed, timed-out or cancelled state was found.

- **GNI_RC_INVALID_PARAM**: An invalid `nic_handle`, `remote_addr` or `remote_id` was specified.

- **GNI_RC_NO_MATCH**: No datagram in the completed, timed-out or cancelled state was found.

3.5.15 PostDataProbeById

The `GNI_PostDataProbeById` function returns the `datagram_id` of the first datagram found in completed, timed out, or canceled state for the CDM associated with the input NIC handle. This function should be used for probing for completion of datagrams that were previously posted using the `GNI_EpPostDataWId` function.

This function must be used in conjunction with `GNI_EpPostDataTestById` or `GNI_EpPostDataWaitById` to obtain the data exchanged in the datagram transaction.

3.5.15.1 Synopsis

```c
gni_return_t GNI_PostDataProbeById (  
    IN gni_nic_handle_t nic_handle,  
    OUT uint64_t *datagram_id)  
```
### 3.5.15.2 Parameters

- **nic_handle**: Handle of the NIC that is associated with the CDM for which the datagram status is being probed.

- **datagram_id**: The datagram ID of the first datagram with a datagram ID specified found in the completed, timed-out or cancelled state. Valid only if the return value is GNI_RC_SUCCESS.

### 3.5.15.3 Return Codes

- **GNI_RC_SUCCESS**: A datagram in the completed, timed-out or cancelled state was found.

- **GNI_RC_INVALID_PARAM**: An invalid nic_handle, datagram_id was specified.

- **GNI_RC_NO_MATCH**: No datagram with the specified ID found in the completed, timed-out or cancelled state.

### 3.5.16 PostDataProbeWaitById

The PostDataProbeWaitById function returns the post ID of the first datagram posted with a datagram ID found in completed, timed out, or canceled state for the CDM associated with the input nic_handle. This function must be used in conjunction with gni_ep_postdata_test_by_id or gni_ep_postdata_wait_by_id to obtain data exchanged in the datagram transaction.

#### 3.5.16.1 Synopsis

```c
gni_return_t GNI_PostDataProbeWaitById (  
    IN gni_nic_handle_t nic_handle,  
    IN uint32_t timeout,  
    OUT uint64_t *datagram_id )
```
3.5.16.2 Parameters

- **nic_handle**: Handle of the NIC that is associated with the CDM for which the datagram status is being probed.
- **timeout**: The count (in milliseconds) that this function waits, for transaction to complete. Set to -1 if no timeout is desired.
- **datagram_id**: The first datagram ID found in the completed, timed-out or cancelled state. Valid only if the return value is GNI_RC_SUCCESS.

3.5.16.3 Return Codes

- **GNI_RC_SUCCESS**: A datagram with the specified id was found in the completed, timed-out or cancelled state.
- **GNI_RC_INVALID_PARAM**: An invalid nic_handle, or timeout was specified.
- **GNI_RC_TIMEOUT**: No datagram with a datagram ID specified and in the completed, timed-out or cancelled state was found before the timeout expired.
- **GNI_RC_NO_MATCH**: No datagram with the specified ID found in the completed, timed-out or cancelled state.

3.6 FMA DM

3.6.1 PostFma

The PostFma function executes a data transaction (PUT, GET, or AMO) by storing into the directly mapped FMA window to initiate a series of FMA requests. It returns before the transaction is confirmed by the remote NIC.

Zero-length FMA PUT operations are supported. Zero-length FMA GET and zero-length FMA AMO operations are not supported.

3.6.1.1 Synopsis

```c
gni_return_t GNI_PostFma(
    IN gni_ep_handle_t ep_handle,
    IN gni_post_descriptor_t *post_descr)
```
3.6.1.2 Parameters

- `ep_handle` Instance of a local endpoint.
- `post_descr` Pointer to a descriptor to be posted.

3.6.1.3 Return Codes

- **GNI_RC_SUCCESS**
  - The descriptor was successfully posted.
- **GNI_RC_INVALID_PARAM**
  - The endpoint handle was invalid.
- **GNI_RC_ALIGNMENT_ERROR**
  - The posted source or destination data pointers or data length are not properly aligned. There are no alignment restrictions on PUTs. GETs require 4 byte-alignment. AMOs require 8 byte-alignment, except AAX which requires 16 byte-alignment.
- **GNI_RC_ERROR_RESOURCE**
  - The transaction request failed due to insufficient resources.

3.7 FMA Short Messaging (SMSG)

3.7.1 SmsgInit

The SmsgInit function configures the short messaging protocol on the given endpoint. Short messaging buffers must be zeroed before calling SmsgInit.

3.7.1.1 Synopsis

```c
gni_return_t GNI_SmsgInit (
    IN gni_ep_handle_t ep_handle,
    IN gni_smsg_attr_t *local_smsg_attr,
    IN gni_smsg_attr_t *remote_smsg_attr)
```
3.7.1.2 Parameters

*ep_handle*  
Instance of an endpoint.

*local_smsg_attr*  
Pointer to a list of local parameters used for short messaging. Parameter values are defined using the `gni_smsg_attr` structure.

*remote_smsg_attr*  
Pointer to a list of remote parameters that are used for short messaging, provided by a peer. Parameter values are defined using the `gni_smsg_attr` structure.

3.7.1.3 Return Codes

**GNI_RC_SUCCESS**  
Operation completed successfully.

**GNI_RC_INVALID_PARAM**  
One of the input parameters was invalid.

**GNI_RC_INVALID_STATE**  
Endpoint is not bound.

**GNI_RC_ERROR_NOMEM**  
Insufficient memory to allocate short message internal structures.

3.7.2 SmsgSend

The `SmsgSend` function sends a message to the remote peer by copying it into the preallocated remote buffer space, using the FMA mechanism. The function returns before the delivery is confirmed by the remote NIC. When the endpoint is set with the `GNI_SMSG_TYPE_MBOX_AUTO_RETRANSMIT` flag, the system attempts to retransmit messages when certain transaction failures occur. This `SmsgSend` function is a non-blocking call.
The number of GNI SMSG send credits equates to the maximum number of messages the SMSG mailbox can hold at one time. Send credits are independent of buffer space available, so when sending messages smaller than the max message size specified to GNI_SmsgInit, the mailbox space is underutilized. Each SMSG connection will set the SMSG send credits count equal to the smsg_q_sz by default. See gni_msgq_attr on page 127.

Note: The SMSG interface uses the FMA mechanism with adaptive routing. This allows SMSG messages to arrive out of order at the target node. Therefore, it is possible for completion events to be delivered to the remote completion queue while GNI_SmsgGetNext reports that no new messages are available. To handle this case when using remote events to detect the arrival of SMSG sends, be sure to clear all messages from an endpoint using GNI_SmsgGetNext after receiving each remote completion event.

### 3.7.2.1 Synopsis

```c
gni_return_t GNI_SmsgSend(
    IN gni_ep_handle_t ep_handle,
    IN void *header,
    IN uint32_t header_length,
    IN void *data,
    IN uint32_t data_length,
    IN uint32_t *msg_id)
```

### 3.7.2.2 Parameters

- **ep_handle**  
  An instance of an endpoint.

- **header**  
  A pointer to the header of a message.

- **header_length**  
  The length of the header in bytes.

- **data**  
  A pointer to the payload of the message.

- **data_length**  
  The length of the payload in bytes.

- **msg_id**  
  Identifier for application to track transaction. Only valid for short messaging using GNI_SMSG_TYPE_MBOX_AUTO_RETRANSMIT type, otherwise ignored.
3.7.2.3 Return Codes

GNI_RC_SUCCESS

The transmission has been initiated.

GNI_RC_INVALID_PARAM

The endpoint handle was invalid or the endpoint is not initialized for short messaging.

GNI_RC_NOT_DONE

No credits available to send the message.

GNI_RC_ERROR_RESOURCE

The operation failed due to insufficient memory.

3.7.3 SmsgSendWTag

The SmsgSendWTag function sends a tagged message to the remote peer, by copying it into the preallocated remote buffer space, using the FMA mechanism. The function returns before the delivery is confirmed by the remote NIC. When the endpoint is set with GNI_SMSG_MBOX_AUTO_RETRANSMIT type, the system attempts to retransmit for certain transaction failures. This is a non-blocking call.

3.7.3.1 Synopsis

```c
gni_return_t GNI_SmsgSendWTag(
    IN gni_ep_handle_t ep_hndl,
    IN void *header,
    IN uint32_t header_length,
    IN void *data,
    IN uint32_t data_length,
    IN uint32_t msg_id,
    IN uint8_t tag)
```
3.7.3.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ep_hndl</td>
<td>An instance of an endpoint.</td>
</tr>
<tr>
<td>header</td>
<td>A pointer to the header of a message.</td>
</tr>
<tr>
<td>header_length</td>
<td>The length of the header in bytes.</td>
</tr>
<tr>
<td>data</td>
<td>A pointer to the payload of the message.</td>
</tr>
<tr>
<td>data_length</td>
<td>The length of the payload in bytes.</td>
</tr>
<tr>
<td>msg_id</td>
<td>Identifier for application to track transaction. Only valid for short messaging using GNI_SMSG_TYPE_MBOX_AUTO_RETRANSMIT type, otherwise ignored.</td>
</tr>
<tr>
<td>tag</td>
<td>Tag associated with the short message.</td>
</tr>
</tbody>
</table>

3.7.3.3 Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNI_RC_SUCCESS</td>
<td>The transmission was initiated.</td>
</tr>
<tr>
<td>GNI_RC_INVALID_PARAM</td>
<td>The endpoint handle was invalid or the endpoint is not initialized for short messaging.</td>
</tr>
<tr>
<td>GNI_RC_NOT_DONE</td>
<td>No credits available to send the message.</td>
</tr>
<tr>
<td>GNI_RC_ERROR_RESOURCE</td>
<td>The operation failed due to insufficient memory.</td>
</tr>
</tbody>
</table>

3.7.4 SmsgGetNext

The SmsgGetNext function returns a pointer to the header of the newly arrived message and makes this message current. You can set up your application to copy the message out of the mailbox or process it immediately. This is a non-blocking call.

3.7.4.1 Synopsis

```c
gni_return_t GNI_SmsgGetNext (  
    IN gni_ep_handle_t ep_handle,  
    OUT void **header)  
```
3.7.4.2 Parameters

- `ep_handle` Instance of an endpoint.
- `header` Pointer to the header of the newly arrived message.

3.7.4.3 Return Codes

- **GNI_RC_SUCCESS**
  The new message arrived successfully.

- **GNI_RC_INVALID_PARAM**
  The endpoint handle was invalid or the endpoint is not initialized for short messaging.

- **GNI_RC_NOT_DONE**
  There are no new messages available.

3.7.5 SmsgGetNextWTag

The SmsgGetNextWTag function returns a pointer to the header of the newly arrived message and makes this message current if the input tag matches the tag of the newly arrived message. An application may decide to copy the message header out of the mailbox or process the header immediately. This is a non-blocking call.

3.7.5.1 Synopsis

```c
gni_return_t GNI_SmsgGetNextWTag (  
  IN gni_ep_handle_t ep_hndl,  
  OUT void **header,  
  INOUT uint8_t *tag)
```

3.7.5.2 Parameters

- `ep_handle` Instance of an endpoint.
- `header` Pointer to the header of the newly arrived message.
- `tag` On input, a pointer to the value of the remote event to be matched. A wildcard value of GNI_SMSG_ANY_TAG is used to match any tag value of the incoming message. The value is set to that of the matching remote event on output.
3.7.5.3 Return Codes

GNI_RC_SUCCESS
The new message arrived successfully.

GNI_RC_INVALID_PARAM
The endpoint handle was invalid or the endpoint is not initialized for short messaging.

GNI_RC_NOT_DONE
There are no new messages available.

GNI_RC_NO_MATCH
The message is available, but the tag of the message does not match the value supplied in the tag argument.

3.7.6 SmsgRelease

The SmsgRelease function releases the current message buffer. It must be called only after GetNext has returned GNI_RC_SUCCESS. This is a non-blocking call. The message returned by the GetNext function must be copied out or processed before making this call.

3.7.6.1 Synopsis

gni_return_t GNI_SmsgRelease (  
   IN gni_ep_handle_t ep_handle)

3.7.6.2 Parameters

ep_handle Instance of an endpoint.

3.7.6.3 Return Codes

GNI_RC_SUCCESS
The current message is successfully released.

GNI_RC_INVALID_PARAM
The endpoint handle was invalid or the endpoint is not initialized for short messaging.

GNI_RC_NOT_DONE
There is no current message. The GetNext function must return GNI_RC_SUCCESS before calling this function.
3.7.7 SmsgSetMaxRetrans

Configure SMSG maximum retransmit count. End-points associated with the NIC handle provided will give up retransmitting SMSG transactions and return GNI_RC_TRANSACION_ERROR when the retransmit count has been reached.

3.7.7.1 Synopsis

```c
gni_return_t GNI_SmsgSetMaxRetrans (
    IN gni_nic_handle_t nic_handle
    IN uint16_t max_retrans);
```

3.7.7.2 Parameters

- `nic_handle` NIC handle to change.
- `max_retrans` Maximum retransmit count.

3.7.7.3 Return Codes

- **GNI_RC_SUCCESS**
  
  The current message is successfully released.

- **GNI_RC_INVALID_PARAM**
  
  The endpoint handle was invalid or the endpoint is not initialized for short messaging.

- **GNI_RC_NOT_DONE**
  
  There is no current message. The GetNext function must return GNI_RC_SUCCESS before calling this function.
3.8 Shared Message Queue (MSGQ)

The MSGQ permits applications based on programming models which exchange short control messages, such as MPICH2, to achieve greater scalability. To create a shared message queue system, each job instance in a job must:

1. Create and fill a gni_msgq_attr_t structure.
2. Call GNI_MsgqInit to attach to a shared message queue.
3. Perform a global barrier.

The leader instance is the one that first attempts to attach to the shared message queue and thereby creates the shared buffer resources. Any one instance on each node can act as the leader and perform the following steps:

1. Call GNI_MsgqGetConnAttrs with the message queue handle obtained during initialization, the remote node's PE address and a pointer to a gni_msgq_ep_attr_t structure. This fills the gni_msgq_ep_attr_t structure with the attributes needed on the remote node to establish the internode connection.
2. Trade the gni_msgq_ep_attr_t structure with the node with the PE address specified in previous step.
3. Call GNI_MsgqConnect with the received endpoint attribute structure.
4. Repeat from step 1 for each remote node participating in the message queue system.
5. Perform a global barrier.

Job instances will send messages using the GNI_MsgqSend function. If successful this function delivers a completion event to the completion queue attached to the endpoint provided in the send. Job instances will receive messages using the GNI_MsgqProgress function.

3.8.1 MsgqInit

Create and initialize the resources required for the shared message queue, including the shared memory buffer and the receive completion queue. Receive CQ's will be created per job instance on each node.

Each job instance will attach to the shared buffer during initialization. The first job instance to attach to the shared buffer will create shared buffer resources and initialize all shared variables according to initialization attributes described in gni_msgq_attr on page 127.

Register the shared region with a private receive completion queue and store the provided message queue attributes as control information in the shared area. Map connection information stored in the shared buffer to its remote PE address.
Job attributes must be equal for all instances in the job and node attributes must be equal for each instance on a node. Instance attributes may be unique for each job instance.

The number of connections that the shared message queue will support will be specified at initialization time. Once all job instances complete their call to GNI_MsgqInit, each node assigns one job instance as the leader, responsible for each remote node connection in the message queue.

### 3.8.1.1 Synopsis

```c
GNI_MsgqInit (  
  IN gni_nic_handle_t nic_handle,  
  IN gni_msgq_rcv_cb_func *rcv_cb,  
  IN void *cb_data,  
  IN gni_cq_handle_t snd_cq,  
  IN gni_msgq_attr_t *attrs,  
  OUT gni_msgq_handle_t *msgq_handle)
```

### 3.8.1.2 Parameters

- **nic_handle**  
  The handle of the NIC device to attach to shared message queue.

- **rcv_cb**  
  Callback function to handle received messages.

- **cb_data**  
  User data to pass to the receive callback function specified by `rcv_cb`.

- **snd_cq**  
  Send completion queue to use with message queue.

- **attrs**  
  Attributes for message queue initialization.

- **msgq_handle**  
  A handle for the created message queue resources.
3.8.1.3 Return Codes

GNI_RC_SUCCESS
Message Queue initialization succeeded.

GNI_RC_INVALID_PARAM
One of the input parameters was invalid.

GNI_RC_ERROR_NOMEM
There was insufficient memory available to attach to the shared memory region.

GNI_RC_INVALID_STATE
The attributes provided do not match the existing message queue attributes or all instances were not ready to attach to the shared memory area.

GNI_RC_PERMISSION_ERROR
The hugetlbfs file system was not available.

3.8.2 MsgqRelease

Releases all resources created during GNI_MsgqInit. All transactions must be completed (or all end-points destroyed) before calling GNI_MsgqRelease. The user is unable to allocate new connections while this function is executing. Each connection is locked and disabled unless there are outstanding transactions on that connection. If there are outstanding transactions, returns GNI_RC_NOT_DONE.

3.8.2.1 Synopsis

GNI_MsgqRelease (
    IN gni_msgq_handle_t msgq_hndl)

3.8.2.2 Parameters

msgq_hndl       The handle for the message queue to use for the operation.
3.8.2.3 Return Codes

GNI_RC_SUCCESS
Message Queue resources successfully released.

GNI_RC_INVALID_PARAM
One of the input parameters was invalid.

GNI_RC_NOT_DONE
There are outstanding message queue transactions; those connections are not released, leaving a partially disabled message queue.

3.8.3 MsgqGetConnAttrs

Assigns connection resources to a remote end-point address and returns attributes for completing the connection. The remote PE address provided is assigned to an SMSG control structure and mailbox for use in an internode connection. The attributes must be traded with the remote end-point to establish the connection.

3.8.3.1 Synopsis

GNI_MsgqGetConnAttrs (  
    IN gni_msgq_handle_t msgq_hndl,  
    IN uint32_t pe_addr,  
    OUT gni_msgq_ep_attr_t *attrs,  
    OUT uint32_t *attrs_size)  

3.8.3.2 Parameters

msgq_hndl The handle of the message queue to use for the operation.

pe_addr The PE address of the remote end-point to assign connection resources to. If NTT is enabled, it is the virtual address.

attrs The attribute structure needed to establish a message queue connection on the remote end-point.

attrs_size If non-null, returns the size of the attrs structure.
### 3.8.3.3 Return Codes

- **GNI_RC_SUCCESS**
  
  Connection resources were assigned to the PE address.

- **GNI_RC_INVALID_PARAM**
  
  One of the input parameters was invalid.

- **GNI_RC_INVALID_STATE**
  
  Connection resources have already been assigned to the PE address provided.

- **GNI_RC_ERROR_RESOURCE**
  
  All connection resources have already been assigned.

- **GNI_RC_PERMISSION_ERROR**
  
  Message queue initialization has not completed or `MsgqRelease` has been started.

### 3.8.4 **MsgqConnect**

Connect to a shared message queue. Use the remote PE address, `pe_addr` to look up the shared connection resources that were assigned during `GNI_MsgqGetConnAttrs`. Complete the internode message queue connection by adding the remote end-point attributes `attrs`, to the connection resources.

#### 3.8.4.1 Synopsis

```c
GNI_MsgqConnect (  
    IN gni_msgq_handle_t msgq_handle,  
    IN uint32_t pe_addr,  
    IN gni_msgq_ep_attr_t *attrs)
```

#### 3.8.4.2 Parameters

- **msgq_handle**
  
  The handle of the message queue to use for the operation.

- **pe_addr**
  
  The PE address of the remote end-point to assign connection resources to. If NTT is enabled, it is the virtual address.

- **attrs**
  
  The attribute structure received from the remote node.
3.8.4.3 Return Codes

GNI_RC_SUCCESS
The connection was established.

GNI_RC_INVALID_PARAM
One of the input parameters was invalid.

GNI_RC_NO_MATCH
The associated connection resources could not be found.

GNI_RC_INVALID_STATE
A connection to the PE specified by the attribute structure has already been established.

GNI_RC_PERMISSION_ERROR
Message queue initialization has not completed or MsgqRelease has been started.

3.8.5 MsgqConnRelease

Release connection resources from a remote PE, assigned via GNI_MsgqGetConnAttrs. All outstanding transactions on the connection must be completed before calling GNI_MsgqConnRelease. Connection resources released in this call may be reassigned with a call to GNI_MsgqGetConnAttrs.

3.8.5.1 Synopsis

GNI_MsgqConnRelease (  
    IN gni_msgq_handle_t msgq_handle,  
    IN uint32_t pe_addr)

3.8.5.2 Parameters

msgq_handle
The handle of the message queue to use for the operation.

pe_addr
The PE address of the remote end-point to release.
3.8.5.3 Return Codes

GNI_RC_SUCCESS

The connection resources were released.

GNI_RC_INVALID_PARAM

One of the input parameters was invalid.

GNI_RC_NO_MATCH

The associated message queue connection for the specified PE could not be found.

GNI_RC_NOT_DONE

There are outstanding message queue transactions; the connection was not released.

3.8.6 MsgqSend

Send a message after using the specified end-point handle to look up a message queue connection and the target information. Use the remote address in the endpoint to look up the associated message queue connection resources for send operations.

If successful, deliver a send completion event to the completion queue attached to the endpoint provided in the send. Event data will have type GNI_CQ_EVENT_TYPE_MSGQ. If the send was successful, the lower 32 bits of the event data will contain the message ID used for the send. If the event has an error status, the lower 32 bits of the event data will contain the remote instance ID in the send endpoint. This matches the behavior of GNI_SmsgSend.

3.8.6.1 Synopsis

```c
gni_return_t GNI_MsgqSend (  
    IN gni_msgq_handle_t msgq_handle,  
    IN gni_ep_handle_t ep_handle,  
    IN void *header,  
    IN uint32_t header_length,  
    IN void *msg,  
    IN uint32_t msg_length,  
    IN uint32_t msg_id  
    IN uint8_t msg_tag)
```
3.8.6.2 Parameters

msgq_handle

The handle for the message queue to use for the operation.

ep_handle

The endpoint describing the target for the send.

header

A pointer to the header of a message.

header_length

The length of the header in bytes.

msg

A pointer to the message.

msg_length

The length of the message in bytes.

msg_id

Message identifier. Returned in a local completion event.

msg_tag

The message tag sent with message data.

3.8.6.3 Return Codes

GNI_RC_SUCCESS

The send completed successfully.

GNI_RC_INVALID_PARAM

An invalid input parameter was provided.

GNI_RC_NO_MATCH

No message queue connection for the end-point was found.

GNI_RC_NOT_DONE

No credits available to send the message.

GNI_RC_SIZE_ERROR

The message size exceeds the maximum message size.

GNI_RC_INVALID_STATE

Connection resources exist but are inactive (not yet connected).

3.8.7 MsgqSize

Returns the size of the MSGQ allocated shared buffer given a set of initialization parameters. The size is specified in bytes. The size is always a multiple of the configured hugetlbfs hugepage size.
3.8.7.1 Synopsis

GNI_MsgqSize (
    IN gni_msgq_attr_t *attrs,
    OUT uint32_t *size)

3.8.7.2 Parameters

attrs    The attributes for the message queue system initialization.

size    The size, in bytes, required to create the MSGQ with the given set of parameters.

3.8.7.3 Return Codes

GNI_RC_SUCCESS

The operation completed successfully.

GNI_RC_INVALID_PARAM

The input parameter was invalid.

3.8.8 MsgqProgress

Polls the receive completion queue until an event is received or the timeout expires. If (-1) is specified as the timeout value, the function will return immediately if there are no completion events ready.

When an event is received, the registered receive callback function is called with the message data. If the user provided callback function returns true, GNI_MsgqProgress will attempt to process another message. If the callback returns false, GNI_MsgqProgress will return immediately.

Each job instance on a node will have registered the shared buffer with the receive CQ at initialization.

3.8.8.1 Synopsis

GNI_MsgqProgress (
    IN gni_msgq_handle_t msgq_handle,
    IN uint32_t timeout)

3.8.8.2 Parameters

msgq_handle    The handle of the message queue to use for the operation.

timeout    The number of milliseconds to block waiting for each message.
### 3.8.8.3 Return Codes

- **GNI_RC_SUCCESS**
  
  All messages were processed.

- **GNI_RC_INVALID_PARAM**
  
  Invalid `msgq_handle`.

- **GNI_RC_NOT_DONE**
  
  Messages may still be available for processing.

- **GNI_RC_ERROR_RESOURCE**
  
  The shared message queue is blocked by another instance or the send CQ is full.

- **GNI_RC_INVALID_STATE**
  
  An unexpected completion queue event was received.

- **GNI_RC_ERROR_NOMEM**
  
  Insufficient memory was available to complete the operation.

### 3.9 RDMA (BTE)

#### 3.9.1 PostRdma

The `PostRdma` function adds a descriptor to the tail of the RDMA queue and returns immediately.

#### 3.9.1.1 Synopsis

```c
gni_return_t GNI_PostRdma (
    IN gni_ep_handle_t ep_handle,
    IN gni_post_descriptor_t *post_descr)
```

#### 3.9.1.2 Parameters

- **ep_handle**
  
  Instance of a local endpoint.

- **post_descr**
  
  Pointer to the descriptor to be posted to the queue.
3.9.1.3 Return Codes

GNI_RC_SUCCESS
The descriptor was successfully posted.

GNI_RC_INVALID_PARAM
The endpoint handle was invalid.

GNI_RC_ALIGNMENT_ERROR
Posted source, destination data pointers, or data length are not properly aligned.

GNI_RC_ERROR_RESOURCE
The transaction request could not be posted due to insufficient resources.

GNI_RC_ERROR_NOMEM
Insufficient memory to complete the operation.

GNI_RC_PERMISSION_ERROR
The user's buffer R/W permissions conflict with the access type.

3.10 Completion Queue Processing

A uGNI application monitors the completion queue in order to track events, process event data and handle errors.

The CqGetEvent, CqWaitEvent, and CqVectorWaitEvent functions return a completion queue entry which may be used as input to event processing functions and error decoding functions. See gni_cq_entry on page 119 for more detailed information about completion queue entries.

3.10.1 CqTestEvent

The CqTestEvent function monitors the specified completion queue for a completion entry. If a completion entry is found, it returns GNI_RC_SUCCESS, unless the CQ is full, in which case it returns GNI_RC_ERROR_RESOURCE. If no completion entry is found, GNI_RC_NOT_DONE is returned. No processing of new entries is performed by this function.

3.10.1.1 Synopsis

GNI_CqTestEvent (
    IN gni_cq_handle_t cq_handle)
3.10.1.2 Parameters

\textit{cq\_handle} \quad The handle for the completion queue.

3.10.1.3 Return Codes

\begin{itemize}
\item \textbf{GNI\_RC\_SUCCESS} \\
A completion entry was found on the completion queue.
\item \textbf{GNI\_RC\_NOT\_DONE} \\
No new completion entries are on the completion queue.
\item \textbf{GNI\_RC\_INVALID\_PARAM} \\
The completion queue handle was invalid.
\item \textbf{GNI\_RC\_ERROR\_RESOURCE} \\
CQ is full and CQ entries may have been lost.
\end{itemize}

3.10.2 CqGetEvent

The \texttt{CqGetEvent} function returns information about the next event by polling the specified completion queue for a completion entry. If a completion entry is found, it returns the event data stored in the entry. \texttt{CqGetEvent} is a non-blocking call.

The calling process must subsequently invoke the appropriate function to dequeue the completed descriptor. \texttt{CqGetEvent} only de-queues the completion entry from the completion queue.

3.10.2.1 Synopsis

\begin{verbatim}
gni_return_t GNI_CqGetEvent (  
   IN gni_cq_handle_t cq_handle,  
   OUT gni_cq_entry_t *event_data)
\end{verbatim}

3.10.2.2 Parameters

\begin{itemize}
\item \textit{cq\_handle} \quad The handle for the completion queue.
\item \textit{event\_data} \quad Event entry data is placed at the address pointed to by \textit{event\_data}. The contents of an event entry is dictated by the status and type of the transaction associated with the event. See \texttt{gni\_cq\_entry} on page 119.
\end{itemize}
3.10.2.3 Return Codes

GNI_RC_SUCCESS
A completion entry was found in the completion queue.

GNI_RC_NOT_DONE
No new completion entries are in the completion queue.

GNI_RC_INVALID_PARAM
The completion queue handle was invalid.

GNI_RC_ERROR_RESOURCE
The completion queue is in an overrun (full) state and completion queue events may have been lost.

GNI_RC_TRANSACTION_ERROR
A network error was encountered while processing a transaction.

3.10.3 CqWaitEvent

The CqWaitEvent function polls the specified completion queue for a completion entry. If CqWaitEvent finds a completion entry, it immediately returns event data. If no completion entry is found, the caller is blocked until a completion entry is generated, or until the timeout value expires. The completion queue must be created with the GNI_CQ_BLOCKING mode set in order to be able to block on it.

3.10.3.1 Synopsis

gni_return_t GNI_CqWaitEvent(
    IN gni_cq_handle_t cq_handle,
    IN uint64_t timeout,
    OUT gni_cq_entry_t *event_data)

3.10.3.2 Parameters

cq_handle  The handle for the completion queue.

timeout    The number of milliseconds to block before returning to the caller; set this to -1 if no timeout is desired.

event_data Event entry data is always placed at the address pointed to by event_data. The contents of an event entry is dictated by the status and type of the transaction associated with the event. See gni_cq_entry on page 119.
3.10.3.3 Return Codes

GNI_RC_SUCCESS
A completion entry was found on the completion queue.

GNI_RC_TIMEOUT
The request timed out and no completion entry was found.

GNI_RC_INVALID_PARAM
The completion queue handle was invalid.

GNI_RC_ERRORRESOURCE
The completion queue was not created in the GNI_CQ_BLOCKING mode.

GNI_RC_TRANSACTION_ERROR
A network error was encountered while processing a transaction.

3.10.4 CqVectorWaitEvent

The CqVectorWaitEvent function polls the specified completion queues for a completion entry. If CqVectorWaitEvent finds a completion entry, it immediately returns event data.

If no completion entry is found, the caller is blocked until a completion entry is generated, or until the timeout value expires. The completion queues must be created with the GNI_CQ_BLOCKING mode set in order to be able to block on it.

3.10.4.1 Synopsis

\[
gni_return_t \ GNI\_CqVectorWaitEvent (\ 
\IN \ gni\_cq\_handle_t *cq\_handls,\ 
\IN \ uint32_t \ num\_cqs\ 
\IN \ uint64_t \ timeout,\ 
\OUT \ gni\_cq\_entry_t *event\_data,\ 
\OUT \ uint32_t *which)\]

3.10.4.2 Parameters

\begin{itemize}
  \item \textit{cq\_handls} \quad \text{Array of handles for the completion queues.}
  \item \textit{num\_cqs} \quad \text{Number of completion queue handles.}
  \item \textit{timeout} \quad \text{The number of milliseconds to block before returning to the caller; set this to \texttt{-1} if no timeout is desired.}
\end{itemize}
event_data: Event entry data is always placed at the address pointed to by event_data. The contents of an event entry is dictated by the status and type of the transaction associated with the event. See gni_cq_entry on page 119.

which: Returns the index of the CQ within the cq_handls array which returned event_data on success. Undefined otherwise.

### 3.10.4.3 Return Codes

- **GNI_RC_SUCCESS**
  A completion entry was found on the completion queue.

- **GNI_RC_TIMEOUT**
  The request timed out and no completion entry was found.

- **GNI_RC_INVALID_PARAM**
  One of the completion queue handles was invalid.

- **GNI_RC_ERROR_RESOURCE**
  One of the completion queues was not created in the GNI_CQ_BLOCKING mode.

- **GNI_RC_TRANSACTION_ERROR**
  A network error was encountered while processing a transaction.

### 3.10.5 GetCompleted

The GetCompleted function gets the next completed post descriptor from the specified completion queue. The descriptor is removed from the head of the queue and the address of the descriptor is returned.

A GNI_RC_DESCRIPTOR_ERROR is returned if the transaction has failed and the error was reported by the CqGetEvent function in the event_data parameter. In this case, the error information from the event_data is copied to the status field of the descriptor. GetCompleted is a non-blocking call.

#### 3.10.5.1 Synopsis

```c
gni_return_t GNI_GetCompleted (  
    IN gni_cq_handle_t cq_handle,  
    IN gni_cq_entry_t event_data,  
    OUT gni_post_descriptor_t **post_descr)
```
3.10.5.2 Parameters

- **cq_handle**: Handle for the completion queue.
- **event_data**: The event returned by the CqGetEvent function.
- **post_desc**: Returns a pointer to the address of the descriptor that has completed.

3.10.5.3 Return Codes

- **GNI_RC_SUCCESS**: A completed descriptor was returned with a successful completion status.
- **GNI_RC_DESCRIPTOR_ERROR**: If the corresponding post queue (FMA, RDMA or AMO) is empty, the descriptor pointer is set to NULL, otherwise, a completed descriptor is returned with an error completion status.
- **GNI_RC_INVALID_PARAM**: The CQ handle was invalid.
- **GNI_RC_TRANSACTION_ERROR**: A completed descriptor was returned with a network error status.

3.10.6 PostCqWrite

The PostCqWrite function executes a CQ write transaction to a remote CQ. It returns before the transaction is confirmed by the remote NIC.

3.10.6.1 Synopsis

```c
gni_return_t GNI_PostCqWrite (
    IN gni_ep_handle_t ep_handle,
    IN gni_post_descriptor_t *post_descr)
```

3.10.6.2 Parameters

- **ep_handle**: Instance of a local endpoint.
- **post_descr**: Pointer to a descriptor to be posted.
3.10.6.3 Return Codes

GNI_RC_SUCCESS

The descriptor was successfully posted.

GNI_RC_INVALID_PARAM

The endpoint handle was invalid.

GNI_RC_RESOURCE_ERROR

Insufficient were resources available to initialize the endpoint.

3.10.7 CqErrorStr

The CqErrorStr function decodes the error status encoded in a CQ entry by the hardware.

3.10.7.1 Synopsis

\[
gni_return_t \text{GNI\_CqErrorStr} (\text{IN gni\_cq\_entry\_t entry, OUT void *buffer, IN uint32\_t length})
\]

3.10.7.2 Parameters

- **entry**: CQ entry with error status to decode.
- **buffer**: Pointer to the buffer where the error code is returned.
- **length**: Length of the buffer in bytes.

3.10.7.3 Return Codes

GNI_RC_SUCCESS

The completion queue was successfully destroyed.

GNI_RC_INVALID_PARAM

The cq_handle was invalid.

GNI_RC_SIZE_ERROR

The supplied buffer is too small to contain the error code.

3.10.8 CqErrorRecoverable

The CqErrorRecoverable function translates any error status, encoded by the hardware in a completion queue entry, into a recoverable or unrecoverable flag for application usage.
3.10.8.1 Synopsis

```c
gni_return_t GNI_CqErrorRecoverable (
    IN gni_cq_entry_t cq_handle,
    OUT uint32_t *recoverable)
```

3.10.8.2 Parameters

- `entry` Completion queue entry with error status to be decoded.
- `recoverable` Pointer to the integer flag that will contain the decoded result.

3.10.8.3 Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNI_RC_SUCCESS</td>
<td>The entry was successfully decoded.</td>
</tr>
<tr>
<td>GNI_RC_INVALID_PARAM</td>
<td>Invalid input parameter.</td>
</tr>
<tr>
<td>GNI_RC_INVALID_STATE</td>
<td>The completion queue entry translates to an undefined state.</td>
</tr>
</tbody>
</table>

3.11 Error Handling

3.11.1 SubscribeErrors

The `SubscribeErrors` function creates an error event queue. When this function returns, events start reporting immediately. The error mask, `mask`, determines which errors are reported. See `gni_error_mask` on page 118.

Privileged users, such as superusers, can pass in NULL for `nic_handle` which causes the passed in `device_id` to be used instead. This allows privileged users to subscribe to errors without a CDM being attached. By default, if no `nic_handle` is passed in, then errors are captured for all `ptags`.

3.11.1.1 Synopsis

```c
gni_return_t GNI_SubscribeErrors(
    IN gni_nic_handle_t nic_handle,
    IN uint32_t device_id,
    IN gni_error_mask_t mask,
    IN uint32_t EEQ_size,
    OUT gni_err_handle_t *err_handle)
```
3.11.1.2 Parameters

- **nic_handle**: The handle of the associated NIC.
- **device_id**: The device identifier, for privileged mode (when NULL is passed in for `nic_handle`).
- **mask**: The error mask with corresponding bits set for notification.
- **EEQ_size**: Size of the EEQ. The queue size uses a default of 64 entries if a value of 0 is passed in.
- **err_handle**: This handle is returned to identify the instance in uGNI.

3.11.1.3 Return Codes

- **GNI_RC_SUCCESS**: The operation completed successfully.
- **GNI_RC_INVALID_PARAM**: One of the input parameters is invalid, or a non-privileged user is trying to subscribe without a communication domain.
- **GNI_RC_NO_MATCH**: Specified `device_id` does not exist.
- **GNI_RC_ERROR_RESOURCE**: The event queue could not be created due to insufficient resources.
- **GNI_RC_ERROR_NOMEM**: Insufficient memory to complete the operation.

3.11.2 ReleaseErrors

The `ReleaseErrors` function releases the error event notification and cleans up the memory resources for the event queue.

3.11.2.1 Synopsis

```c
gni_return_t GNI_ReleaseErrors(
    IN gni_err_handle_t err_handle)
```

3.11.2.2 Parameters

- **err_handle**: The handle of the subscribed error events.
3.11.2.3 Return Codes

**GNI_RC_SUCCESS**
The descriptor was successfully posted.

**GNI_RC_INVALID_PARAM**
One of the input parameters was invalid.

**GNI_RC_NOT_DONE**
A thread is still waiting on the event queue.

3.11.3 GetErrorMask

The `GetErrorMask` function returns the error mask associated with an error handle. The mask determines which error events are delivered. See `gni_error_mask` on page 118.

### 3.11.3.1 Synopsis

```c
gni_return_t GNI_GetErrorMask(
    IN gni_err_handle_t err_handle,
    OUT gni_error_mask_t *mask)
```

### 3.11.3.2 Parameters

- `err_handle` The handle of the subscribed error events.
- `mask` The pointer to copy the mask value to.

### 3.11.3.3 Return Codes

**GNI_RC_SUCCESS**
The descriptor was successfully posted.

**GNI_RC_INVALID_PARAM**
The endpoint handle was invalid.

3.11.4 SetErrorMask

The `SetErrorMask` function sets a new error mask for matching events.

### 3.11.4.1 Synopsis

```c
gni_return_t GNI_SetErrorMask(
    IN gni_err_handle_t err_handle,
    IN gni_error_mask_t mask_in,
    IN gni_error_mask_t *mask_out)
```
3.11.4.2 Parameters

- **err_handle**: The handle of the subscribed error events.
- **mask_in**: The error mask with corresponding bits set for notification.
- **mask_out**: The pointer to copy the preset mask value to.

3.11.4.3 Return Codes

- **GNI_RC_SUCCESS**: The descriptor was successfully posted.
- **GNI_RC_INVALID_PARAM**: The endpoint handle was invalid.

3.11.5 GetErrorEvent

The `GetErrorEvent` function gets an error event, if available.

3.11.5.1 Synopsis

```c
gni_return_t GNI_GetErrorEvent(
    IN gni_err_handle_t err_handle,
    IN gni_error_event_t *event)
```

3.11.5.2 Parameters

- **err_handle**: The handle of the subscribed error events.
- **event**: The pointer to the buffer to copy the event into.

3.11.5.3 Return Codes

- **GNI_RC_SUCCESS**: A completed descriptor was returned with a successful completion status.
- **GNI_INVALID_PARAMETER**: The endpoint handle was invalid.
- **GNI_RC_NOT_DONE**: No event was found in the event queue.
3.11.6 **WaitErrorEvents**

The `WaitErrorEvents` function limits the waiting period when waiting for one event to occur. When that one event is triggered, it delays returning to try and coalesce error events. The `timeout` value is specified in number of milliseconds. The number of events copied are stored in the `num_events` structure.

### 3.11.6.1 Synopsis

```c
gni_return_t GNI_WaitErrorEvents(
    IN gni_err_handle_t err_handle,
    IN gni_error_event_t *events,
    IN uint32_t events_size,
    IN uint32_t timeout,
    OUT uint32_t *num_events)
```

### 3.11.6.2 Parameters

- **err_handle** The handle of the subscribed error events.
- **events** The pointer to an array of events structures that will be filled in on a successful return. This pointer must be a valid memory location since the events will be copied from the EEQ.
- **events_size** The size of the array passed in from the events pointer.
- **timeout** After first event is triggered, time to wait for subsequent events.
- **num_events** The number of events copied into the events buffer.

### 3.11.6.3 Return Codes

- **GNI_RC_SUCCESS** The operation completed successfully.
- **GNI_RC_INVALID_PARAM** One of the input parameters was invalid.
- **GNI_RC_TIMEOUT** The request timed out and the event array was not filled all the way.
- **GNI_RC_NOT_DONE** The wait was interrupted by the system.
- **GNI_RC_PERMISSION_ERROR** The events pointer cannot be written to.
### 3.11.7 SetErrorPtag

The `SetErrorPtag` function sets the protection tag for an error handler. This is a privileged operation.

#### 3.11.7.1 Synopsis

```c
gni_return_t GNI_SetErrorPtag(
    IN gni_err_handle_t err_handle,
    IN uint8_t ptag)
```

#### 3.11.7.2 Parameters

- **err_handle**  
  The handle of the subscribed error events.
- **ptag**  
  The protect tag to set for matching error events.

#### 3.11.7.3 Return Codes

- **GNI_RC_SUCCESS**  
  The descriptor was successfully posted.
- **GNI_RC_INVALID_PARAM**  
  The endpoint handle was invalid.
- **GNI_RC_PERMISSION_ERROR**  
  Only the superuser can set `ptag` to something other than the communication domain.

### 3.12 Other

#### 3.12.1 GetNumLocalDevices

The `GetNumLocalDevices` function returns the number of NIC devices.

#### 3.12.1.1 Synopsis

```c
gni_return_t GNI_GetNumLocalDevices (  
    OUT int *num_devices)
```

#### 3.12.1.2 Parameters

- **num_devices**  
  Number of NICs on node.
3.12.3 Return Codes

GNI_RC_SUCCESS
Number of devices was returned successfully.

GNI_RC_INVALID_PARAM
One or more of the parameters was invalid.

GNI_RC_ERROR_RESOURCE
Gemini support missing from kernel.

3.12.2 GetLocalDeviceIds

The GetLocalDeviceIds function returns an array of local NIC devices.

3.12.2.1 Synopsis

gni_return_t GNI_GetLocalDeviceIds ( 
    IN int len 
    OUT int *device_id)

3.12.2.2 Parameters

len number of entries in device_ids.

device_ids pointer to array of local NIC devices.

3.12.2.3 Return Codes

GNI_RC_SUCCESS
Number of devices was returned successfully.

GNI_RC_INVALID_PARAM
One or more of the parameters was invalid.

GNI_RC_ERROR_RESOURCE
Gemini support missing from kernel.

3.13 Enumerations

3.13.1 gni_cq_mode

The gni_cq_mode enumeration defines the modes of operation to use for the completion queue. The flags from this enumeration are used for the CqCreate function mode parameter.
3.13.1.1 Synopsis

typedef enum gni_cq_mode {
    GNI_CQ_NOBLOCK = 0,
    GNI_CQ_BLOCKING
} gni_cq_mode_t;

3.13.1.2 Constants

GNI_CQ_NOBLOCK
    Indicates that the CQ instance does not need to be configured in the blocking mode.

GNI_CQ_BLOCKING
    Indicates that the CQ instance should be able to operate in the blocking mode.

3.13.2 gni_fma_cmd_type

The gni_fma_cmd_type enumeration defines the various FMA commands. A FMA command is set using the amo_cmd member of the gni_post_descriptor structure, which is used by the GNI_PostRdma, GNI_PostFma, and GNI_GetCompleted functions.
3.13.2.1 Synopsis

typedef enum gni_fma_cmd_type {
    GNI_FMA_GET       = 0x000,
    GNI_FMA_PUT       = 0x100,
    GNI_FMA_PUT_MSG   = 0x110,
    GNI_FMA_ATOMIC_FADD = 0x008,
    GNI_FMA_ATOMIC_FADD_C = 0x018,
    GNI_FMA_ATOMIC_FAND = 0x009,
    GNI_FMA_ATOMIC_FAND_C = 0x019,
    GNI_FMA_ATOMIC_FOR  = 0x00A,
    GNI_FMA_ATOMIC_FOR_C = 0x01A,
    GNI_FMA_ATOMIC_FXOR = 0x00B,
    GNI_FMA_ATOMIC_FXOR_C = 0x01B,
    GNI_FMA_ATOMIC_FAX  = 0x00C,
    GNI_FMA_ATOMIC_FAX_C = 0x01C,
    GNI_FMA_ATOMIC_CSWAP = 0x00D,
    GNI_FMA_ATOMIC_CSWAP_C = 0x01D,
    GNI_FMA_ATOMIC_ADD  = 0x108,
    GNI_FMA_ATOMIC_ADD_C = 0x118,
    GNI_FMA_ATOMIC_AND  = 0x109,
    GNI_FMA_ATOMIC_AND_C = 0x119,
    GNI_FMA_ATOMIC_OR   = 0x10A,
    GNI_FMA_ATOMIC_OR_C = 0x11A,
    GNI_FMA_ATOMIC_XOR  = 0x10B,
    GNI_FMA_ATOMIC_XOR_C = 0x11B,
    GNI_FMA_ATOMIC_AX   = 0x10C,
    GNI_FMA_ATOMIC_AX_C = 0x11C,
} gni_fma_cmd_type_t;

3.13.2.2 Constants

GNI_FMA_GET

Reserved for use by GNI.

GNI_FMA_PUT

Reserved for use by GNI.

GNI_FMA_PUT_MSG

Reserved for use by GNI.

GNI_FMA_ATOMIC_FADD

Indicates an atomic fetch and ADD command.

GNI_FMA_ATOMIC_FADD_C

Indicates a cached atomic fetch and ADD command.

GNI_FMA_ATOMIC_FAND

Indicates an atomic fetch and AND command.
GNI_FMA_ATOMIC_FAND_C
   Indicates a cached atomic fetch and AND command.

GNI_FMA_ATOMIC_FOR
   Indicates an atomic fetch and OR command.

GNI_FMA_ATOMIC_FOR_C
   Indicates a cached atomic fetch and OR command.

GNI_FMA_ATOMIC_FXOR
   Indicates an atomic fetch and XOR command.

GNI_FMA_ATOMIC_FXOR_C
   Indicates a cached atomic fetch and XOR command.

GNI_FMA_ATOMIC_FAX
   Indicates an atomic fetch, AND and XOR command.

GNI_FMA_ATOMIC_FAX_C
   Indicates a cached atomic fetch, AND and XOR command.

GNI_FMA_ATOMIC_CSWAP
   Indicates an atomic compare and swap command.

GNI_FMA_ATOMIC_CSWAP_C
   Indicates a cached atomic compare and swap command.

GNI_FMA_ATOMIC_ADD
   Indicates an atomic ADD command.

GNI_FMA_ATOMIC_ADD_C
   Indicates a cached atomic ADD command.

GNI_FMA_ATOMIC_AND
   Indicates an atomic AND command.

GNI_FMA_ATOMIC_AND_C
   Indicates a cached atomic AND command.

GNI_FMA_ATOMIC_OR
   Indicates an atomic OR command.
GNI_FMA_ATOMIC_OR_C
Indicates a cached atomic OR command.

GNI_FMA_ATOMIC_XOR
Indicates an atomic XOR command.

GNI_FMA_ATOMIC_XOR_C
Indicate a cached atomic XOR command.

GNI_FMA_ATOMIC_AX
Indicates an atomic AND and XOR command.

GNI_FMA_ATOMIC_AX_C
Indicates an cached atomic AND and XOR command.

3.13.3 gni_mem_handle_attr

The gni_mem_handle_attr enumeration defines memory mapping mechanism used by a memory segment associated with a particular memory handle. The MemHndlQueryAttr function tests a memory handle for a specific memory handle attribute.

3.13.3.1 Synopsis

typedef enum gni_mem_handle_attr {
  GNI_MEMHNDL_ATTR_VMDH,
  GNI_MEMHNDL_ATTR_MRT,
  GNI_MEMHNDL_ATTR_GART,
  GNI_MEMHNDL_ATTR_IOMMU,
  GNI_MEMHHDL_ATTR_PCI_IOMMU,
  GNI_MEMHHDL_ATTR_CLONE,
} gni_mem_handle_attr_t;
3.13.3.2 Constants

GNI_MEMHNDL_ATTR_VMDH
Indicates that the memory handle is a virtual domain handle. See Virtual Memory Domain Handles on page 53.

GNI_MEMHNDL_ATTR_MRT
Indicates that the registered memory segment identified by the handle is mapped to local memory using the MRT.

GNI_MEMHNDL_ATTR_GART
Indicates that the memory segment identified by the handle is mapped to local memory using the GART.

GNI_MEMHNDL_ATTR_IOMMU
Indicates that the memory segment identified by the handle is mapped to local memory using the I/O memory management unit.

GNI_MEMHNDL_ATTR_PCI_IOMMU
Indicates that the memory segment identified by the handle is mapped to local memory using the PCI I/O memory management unit.

GNI_MEMHNDL_ATTR_CLONE
Indicates that the memory handle is a clone.

3.13.4 gni_post_state

The gni_post_state enumeration defines the flags for the post state of datagram transactions between the endpoints on a local and a remote peer that are in the same communication domain. A pointer to the post state is returned by the EpPostDataTest and EpPostDataWait functions when testing the success of an EpPostData operation.

3.13.4.1 Synopsis

typedef enum gni_post_state{
    GNI_POST_PENDING,
    GNI_POST_COMPLETED,
    GNI_POST_ERROR,
    GNI_POST_TIMEOUT,
    GNI_POST_TERMINATED,
    GNI_POST_REMOTE_DATA
} gni_post_state_t;
3.13.4.2 Constants

GNI_POST_PENDING
Indicates the post is pending.

GNI_POST_COMPLETED
Indicates that the data exchange completed successfully.

GNI_POST_ERROR
Indicates the post did not complete due to an error.

GNI_POST_TIMEOUT
Indicates the post did not complete and timed out.

GNI_POST_TERMINATED
Indicates the post did not complete because it was terminated.

GNI_POST_REMOTE_DATA
Indicates receipt of the remote data, but the remote peer did not acknowledge getting the data from the local side.

3.13.5 gni_post_type

The gni_post_type enumeration defines the values to use for the post transaction. The constant values for this enumeration are used by the type member of the gni_post_descriptor structure, which is used by the GNI_PostRdma, GNI_PostFma, and GNI_GetCompleted functions.

3.13.5.1 Synopsis

typedef enum gni_post_type {
    GNI_POST_RDMA_PUT = 1,
    GNI_POST_RDMA_GET,
    GNI_POST_FMA_PUT,
    GNI_POST_FMA_PUT_W_SYNCFLAG,
    GNI_POST_FMA_GET,
    GNI_POST_AMO
} gni_post_type_t;
3.13.5.2 Constants

GNI_POST_RDMA_PUT
Indicates an RDMA PUT transaction.

GNI_POST_RDMA_GET
Indicates an RDMA GET transaction.

GNI_POST_FMA_PUT
Indicates an FMA PUT transaction.

GNI_POST_FMA_PUT_W_SYNCFLAG
Indicates an FMA PUT transaction with a synchronization flag.

GNI_POST_FMA_GET
Indicates an FMA GET transaction.

GNI_POST_AMO
Indicates an AMO transaction.

3.13.6 gni_return

The gni_return enumeration defines the values to use for return values.

3.13.6.1 Synopsis

typedef enum gni_return {
   GNI_RC_SUCCESS = 0,
   GNI_RC_NOT_DONE,
   GNI_RC_INVALID_PARAM,
   GNI_RC_ERROR_RESOURCE,
   GNI_RC_TIMEOUT,
   GNI_RC_PERMISSION_ERROR,
   GNI_RC_DESCRIPTOR_ERROR,
   GNI_RC_ALIGNMENT_ERROR,
   GNI_RC_INVALID_STATE,
   GNI_RC_NO_MATCH,
   GNI_RC_SIZE_ERROR,
   GNI_RC_TRANSACTION_ERROR,
   GNI_RC_ILLEGAL_OP
} gni_return_t;

3.13.6.2 Constants

GNI_RC_SUCCESS
The operation was successful.
GNI_RC_NOT_DONE

The operation is not permitted.

GNI_RC_INVALID_PARAM

One or more of the parameters was invalid.

GNI_RC_ERROR_RESOURCE

Typically, this error means that sufficient resources or the correct resources are not available to complete the operation.

GNI_RC_TIMEOUT

The request timed out.

GNI_RC_PERMISSION_ERROR

The process does not have the correct permissions to complete the operation.

GNI_RC_DESCRIPTOR_ERROR

If the corresponding post queue (FMA, RDMA or AMO) is empty, the descriptor pointer is set to NULL, otherwise, a completed descriptor is returned with an error completion status.

GNI_RC_ALIGNMENT_ERROR

Posted source or destination data pointers or data length are not properly aligned.

GNI_RC_INVALID_STATE

The caller attempted to attach a communication domain instance to the Gemini NIC device more than once.

GNI_RC_NO_MATCH

The requested item and available items do not coincide.

GNI_RC_SIZE_ERROR

The supplied buffer is too small to contain the error code.

GNI_RC_TRANSACTION_ERROR

Error in processing post data transaction.

GNI_RC_ILLEGAL_OP

The operation being attempted is illegal.
3.13.7 gni_smsg_type

The gni_smsg_type enumeration defines the values to use for the short messaging type. The constant values for this enumeration are used in the msg_type member of the gni_smsg_attr structure.

3.13.7.1 Synopsis

```c
typedef enum gni_smsg_type {
    GNI_SMSG_TYPE_INVALID = 0,
    GNI_SMSG_TYPE_MBOX,
    GNI_SMSG_TYPE_MBOX_AUTO_RETRANSMIT
} gni_smsg_type_t;
```

3.13.7.2 Constants

- **GNI_SMSG_TYPE_INVALID**
  
  Indicates that the short message type is invalid.

- **GNI_SMSG_TYPE_MBOX**
  
  Indicates the MBOX short messaging type.

- **GNI_SMSG_TYPE_MBOX_AUTO_RETRANSMIT**
  
  Indicates that the system attempts to retransmit the message for certain transaction failures.

3.14 Structures

3.14.1 gni_error_event

3.14.1.1 Synopsis

```c
typedef struct gni_error_event {
    uint16_t error_code;
    uint8_t error_category;
    uint8_t ptag;
    uint32_t serial_number;
    uint64_t timestamp;
    uint64_t info_mmrs[4];
} gni_error_event_t;
```

3.14.1.2 Members

- **error_code**
  
  Identifies the error which caused the event. Used by GNI for problem reporting. Codes will not be interpreted by uGNI user.

- **error_category**
  
  Errors are divided into 6 categories:
• **CRITICAL_ERR**
  Caused by uncorrectable memory errors, an invalid hardware configuration, or other hardware issues. In most cases, future use of the NIC is unreliable and a node reboot may be required.

• **TRANSACTION_ERR**
  Caused by errors in a specific transaction sequence, likely due to a software issue. A node reboot is not required.

• **ADDR_TRANS_ERR**
  There were errors in the node address translation and/or memory address translation for a specific transaction. A node reboot is not required.

• **TRANSIENT_ERR**
  There may be transient issues with network, memory, or resource availability (for example, no free descriptors). Software should often be able to recover from these errors by reissuing the transaction.

• **CORRECTABLE_MEM_ERR**
  Benign from a system perspective, but should be monitored by HSS and accounted for.

• **INFO_ERR**
  An event occurred which is not necessarily an error condition.

  **ptag**
  PTag responsible for error, when applicable.

  **serial_number**
  This is a semi-unique identifier for the error. An application can use this to match errors entered into the HSS logs. However, some OS errors come outside the normal error reporting path, so they will have a zero for a serial number.

  **timestamp**
  Time the error was reported.

  **info_mmrs**
  Some errors gather additional information from other registers in the hardware which may be useful information in problem reports. Not used by the uGNI user.

### 3.14.2 gni_error_mask

The mask value can be a bitwise OR of the error categories as defined by the ERRMASK flags found in gni_pub.h.
3.14.2.1 Synopsis

typedef uint8_t gni_error_mask_t;
#define GNI_ERRMASK_CORRECTABLE_MEMORY (1 << 0)
#define GNI_ERRMASK_CRITICAL (1 << 1)
#define GNI_ERRMASK_TRANSACTION (1 << 2)
#define GNI_ERRMASK_ADDRESS_TRANSLATION (1 << 3)
#define GNI_ERRMASK_TRANSIENT (1 << 4)
#define GNI_ERRMASK_INFORMATIONAL (1 << 5)
#define GNI_ERRMASK_DIAG_ONLY (1 << 6)

3.14.3 gni_cq_entry

The completion queue entry (CQE) contains event type, status, and user data components which may be extracted using macros, which operate on receive CQs associated with registered memory. See Create Completion Queue (CQ) on page 34.

3.14.3.1 Synopsis

typedef uint64_t gni_cq_entry_t;

3.14.3.2 Event types

uGNI sets the event type regardless of the result of the transaction. The GNI_CQ_GET_TYPE() macro returns the type of transaction associated with the provided completion event. The valid event types defined in gni_pub.h are:

#define GNI_CQ_EVENT_TYPE_POST 0x0ULL
#define GNI_CQ_EVENT_TYPE_SMSG 0x1ULL
#define GNI_CQ_EVENT_TYPE_MSGQ 0x3ULL

3.14.3.3 User data

The user data component of a CQ event can be extracted using either the GNI_CQ_GET_INST_ID() or GNI_CQ_GET_MSG_ID() macro, which are functionally equivalent.
The user data component in a local CQ entry is set depending on the event type as follows:

- **GNI_CQ_EVENT_TYPE_POST**
  
  Contains data value from the associated local endpoint. Defaults to the instance ID of receiver's CDM, but can be changed per endpoint through the GNI_EpSetEventData interface.

- **GNI_CQ_EVENT_TYPE_SMSG or GNI_CQ_EVENT_TYPE_MSGQ**
  
  If the transaction was successful, contains the message ID for that SMSG/MSGQ transaction.
  
  If the transaction failed, contains the data value from the associated local endpoint. This value defaults to the instance ID of receiver's CDM, but can be changed per endpoint through the GNI_EpSetEventData interface.

The user data component in a receive CQ entry defaults to the instance ID of the sender's CDM, regardless of the transaction type or result; it can be extracted using GNI_CQ_GET_INST_ID(). This value can be changed per endpoint through the EpSetEventData() interface.

### 3.14.3.4 Status

The GNI_CQ_STATUS_OK() macro returns true if the transaction associated with the provided CQ event was successful.

The GNI_CQ_OVERRUN() macro returns true if the provided CQ event caused the completion queue to become full. This indicates that completion events may have been lost.

### 3.14.4 gni_job_limits

The gni_job_limits structure defines job parameters and limits. This structure is used by the GNI_ConfigureJob function.

#### 3.14.4.1 Synopsis

```c
typedef struct gni_job_limits {
    int32_t mdd_limit;
    int32_t mrt_limit;
    int32_t gart_limit;
    int32_t fma_limit;
    int32_t bte_limit;
    int32_t cq_limit;
    int32_t ntt_ctrl;
    int32_t ntt_base;
    int32_t ntt_size;
} gni_job_limits_t;
```
3.14.4.2 Members

- **mdd_limit**: Number of MDDs associated with the given *ptag*.
- **mrt_limit**: Number of MRT entries used by MDDs with the given *ptag*.
- **gart_limit**: Number of GART entries used by MDDs with the given *ptag*.
- **fma_limit**: Number of FMA descriptors associated with the given *ptag*.
- **bte_limit**: Number of outstanding BTE descriptors with the given source *ptag*.
- **cq_limit**: Number of CQ descriptors associated with the given *ptag*.
- **ntt_ctrl**: NTT control flag. The only flag that can be used for this parameter is `GNI_JOB_CTRL_NTT_CLEANUP` which is a directive for the driver to cleanup NTT at the end of the job.
- **ntt_base**: Base entry into NTT.
- **ntt_size**: Size of the NTT.

3.14.5 gni_mem_segment

The `gni_mem_segment` structure defines the address and length of a memory segment. The `MemRegisterSegments` function uses this structure.

3.14.5.1 Synopsis

```c
typedef struct gni_mem_segment {
    uint64_t address;
    uint64_t length;
} gni_mem_segment_t;
```

3.14.5.2 Members

- **address**: Address of the segment.
- **length**: Size of the segment in bytes.

3.14.6 gni_ntt_descriptor

The `gni_ntt_descriptor` structure defines configuration options that can be set in NTT. This structure is used by the `GNI_Create` and `GNI_ConfigureNTT` functions.
3.14.6.1 Synopsis

typedef struct gni_ntt_descriptor {
    uint32_t group_size;
    uint8_t granularity;
    uint32_t *table;
    uint8_t flags;
} gni_ntt_descriptor_t;

3.14.6.2 Members

group_size  Size of the NTT group to configure.

granularity  NTT granularity.

table  Pointer to the array of new NTT values.

flags  Configuration flags.

3.14.7 gni_post_descriptor

The gni_post_descriptor structure defines the transaction descriptors. This structure is used by the GetCompleted, PostFMA, and PostRDMA functions.

3.14.7.1 Synopsis

typedef struct gni_post_descriptor {
    void *next_descr;
    void *prev_descr;
    uint64_t post_id;
    uint64_t status;
    uint16_t cq_mode_complete;
    gni_post_type_t type;
    uint16_t cq_mode;
    uint16_t dlvr_mode;
    uint64_t local_addr;
    gni_mem_handle_t local_mem_hndl;
    uint64_t remote_addr;
    gni_mem_handle_t remote_mem_hndl;
    uint64_t length;
    uint16_t rdma_mode;
    gni_cq_handle_t src_cq_hdl;
    uint64_t sync_flag_value;
    uint64_t sync_flag_addr;
    gni_fma_cmd_type_t amo_cmd;
    uint64_t first_operand;
    uint64_t second_operand;
    uint64_t cqwrite_value;
} gni_post_descriptor_t;
### 3.14.7.2 Members

- **next_descr**: Reserved for use by GNI.
- **prev_descr**: Reserved for use by GNI.
- **post_id**: Reserved for use by GNI.
- **status**: Reserved for use by GNI.
- **cq_mode_complete**: Reserved for use by GNI.
- **type**: Required. The type of transaction. The following types are used for this member:
  - GNI_POST_RDMA_PUT
  - GNI_POST_RDMA_GET
  - GNI_POST_FMA_PUT
  - GNI_POST_FMA_PUT_W_SYNCFLAG
  - GNI_POST_FMA_GET
  - GNI_POST_AMO
  - GNI_POST_CQWRITE

- **cq_mode**: Required. Instructs the Gemini NIC to generate completion events. Only **GNI_CQMODE_GLOBAL_EVENT** and **GNI_CQMODE_REMOTE_EVENT** can be requested for FMA_PUT, FMA_GET and AMO transactions. The following modes are used for this member:
  - **GNI_CQMODE_LOCAL_EVENT**

  Can be used only for BTE transactions, and causes an event to be delivered to the local endpoint’s CQ when the local BTE engine has finished handling that descriptor.

  - **GNI_CQMODE_GLOBAL_EVENT**

  Can be specified for FMA and BTE transactions, and causes an event to be delivered to the local endpoint’s CQ when the data successfully arrives at its destination (either local or remote, depending on the operation).

  - **GNI_CQMODE_REMOTE_EVENT**

  Can be used for FMA and BTE transactions, and causes an event to be delivered to the CQ associated with the remote memory registration when the transaction completes.
• GNI_CQMODE_SILENT
  Generate no completion events to any associated CQ (local or remote).

• GNI_CQMODE_DUAL_EVENTS (  
  GNI_CQMODE_LOCAL_EVENT |  
  GNI_CQMODE_GLOBAL_EVENT )

dlvr_mode  Required. Applications must reset the delivery mode to zero before using a default mode when adaptive routing and hashing are enabled.

• GNI_DLVRMODE_PERFORMANCE
• GNI_DLVRMODE_NO_ADAPT
• GNI_DLVRMODE_NO_HASH
• GNI_DLVRMODE_NO_RADAPT
• GNI_DLVRMODE_IN_ORDER (  
  GNI_DLVRMODE_NO_ADAPT | GNI_DLVRMODE_NO_HASH )

local_addr  Required. The address of the region on the local node. This is the source for PUT and the target for GET operations. It must be a 4-byte aligned for GET operations and 8-byte aligned for AMOs.

local_mem_hndl

  The local memory handle. This member is not required for FMA PUT and AMOs with PUT semantics.

remote_addr

  The address of the remote region. This is the target for PUTs and source for GETs. Must be 4-byte aligned for GET operations and 8-byte aligned for AMOs.

remote_mem_hndl

  Remote memory handle.

length  Number of bytes to move. Must be a multiple of 4-bytes for GETs and multiple of 8-bytes for AMOs.
rdma_mode  There are two modes used for this member:

- **GNI_RDMAMODE_PHYS_ADDR**
  
  If set, the kernel-level application uses a physical address for the local_addr field.

- **GNI_RDMAMODE_FENCE**
  
  If set, causes the completion processing of the transaction descriptor to be delayed until all network responses, associated with the current descriptor as well as all responses associated with previously processed descriptors of the same BTE channel, have been received. Processing of the next descriptor for the channel does not start until the write-back of the current transmit transaction descriptor is issued.

src_cq_hndl

If set, the NIC delivers the source completion events related to this transaction to the specified completion queue instead of the default one.

sync_flag_value

Synchronization value.

sync_flag_addr

Local to deliver synchronization value.

amo_cmd  AMO command for the transaction.

first_operand

First operand required by the AMO command.

second_operand

Second operand required by the AMO command.

cqwrite_value

Value to use for a CQ write. Only six least significant bytes is available to software.

### 3.14.8 gni_smsg_attr

The gni_smsg_attr structure defines the attributes for short messaging. This structure is used by the SmsgInit function.
3.14.8.1 Synopsis

typedef struct gni_smsg_attr {
    gni_smsg_type_t msg_type;
    void *msg_buffer;
    uint32_t buff_size;
    gni_mem_handle_t mem_handle;
    uint32_t mbox_offset;
    uint32_t mbox_maxcredit;
    uint32_t msg_maxsize;
} gni_smsg_attr_t;

3.14.8.2 Members

msg_type  The type of short message buffering method to use. This member uses the following message types:
- GNI_SMSG_TYPE_MBOX
- GNI_SMSG_TYPE_MBOX_AUTO_RETRANSMIT

For both of these types, the buffer space for incoming messages is associated with a single remote endpoint. The GNI_SMSG_TYPE_MBOX_AUTO_RETRANSMIT type supports automatic retransmission of short messages by the GNI library in the event of transient network faults.

msg_buffer A pointer to the beginning of the memory region used for message buffers. Individual message buffers may be associated with different endpoints.

buff_size  Size of the message buffer in bytes for this endpoint.

mem_hdl Memory handle for the memory region used for message buffers.

mbox_offset Offset from msg_buffer in bytes indicating the base address for the message buffer associated with this endpoint.

mbox_maxcredit The maximum number of messages that can be buffered in the message buffer.

msg_maxsize The maximum size of the short message which can be received for this endpoint.

3.14.9 gni_smsg_handle

The gni_smsg_handle structure is reserved for use by the GNI infrastructure.
3.14.10 gni_msgq_attr

The gni_msgq_attr structure defines the attributes for the shared message queue. This structure is used by the MsgqInit function.

3.14.10.1 Synopsis

typedef struct gni_msgq_attr {
    uint32_t      max_msg_sz;
    uint32_t      smsg_q_sz;
    uint32_t      rcv_pool_sz;
    uint32_t      num_msgq_eps;
    uint32_t      nloc_insts;
    uint8_t       modes;
    uint32_t      rcv_cq_sz;
} gni_msgq_attr_t;

3.14.10.2 Members

max_msg_sz  Maximum message size in bytes.

smsg_q_sz  SMSG queue size in units of max_msg_sz. Each SMSG connection sets the SMSG send credits count equal to the smsg_q_sz, by default.

rcv_pool_sz  Number of message buffers in the receive pool.

num_msgq_eps  The number of nodes using the message queue system.

nloc_insts  The number of local instances attaching to the shared message queue on the node.

modes  Mode flags specifying message queue properties.

rcv_q_sz  The number of entries in the receive completion queue.

3.14.11 gni_msgq_rem_inst

3.14.11.1 Synopsis

typedef struct gni_msgq_rem_inst {
    uint32_t id;
    gni_mem_handle_t mdh;
    uint64_t mdh_off;
} gni_msgq_rem_inst_t;
3.14.11.2 Members

- **id**: Instance ID.
- **mdh**: MDH for the SHMEM region.
- **mdh_off**: Offset into the MDH for the SMSG mail box.

3.14.12 gni_msgq_ep_attr

3.14.12.1 Synopsis

```c
struct gni_msgq_ep_attr {
    uint32_t pe_addr;
    uint32_t max_msg_sz;
    uint32_t smsg_q_sz;
    uint32_t num_insts;
    gni_msgq_rem_inst_t insts[GNI_MSGQ_NODE_INSTS_MAX];
} gni_msgq_ep_attr_t;
```

3.14.12.2 Members

- **pe_addr**: The PE address of the remote end-point (virtual, if NTT is enabled).
- **max_msg_sz**: Maximum message size in bytes.
- **smsg_q_sz**: SMSG queue size in units of `max_msg_sz`. Each SMSG connection sets the SMSG send credits count equal to the `smsg_q_sz`, by default.
- **num_insts**: The number of local instances attaching to the shared message queue on the node.
- **insts**: An array of local instance descriptions.
Part II: The DMAPP API
DMAPP is a communication library which supports a logically shared, distributed memory (DM) programming model. DMAPP provides remote memory access (RMA) between processes within a job in a one-sided manner. One-sided remote memory access requests require no active participation by the process at the remote node; synchronization functions may be used to determine when side-effects of locally initiated requests are available.

DMAPP is typically not used directly within user application software. The DMAPP API allows one-sided communication libraries (such as Cray SHMEM), and PGAS compilers (such as Coarray Fortran and UPC), implemented on top of DMAPP, to realize much of the hardware performance of the Gemini based system interconnection network while being reasonably portable to its successors.

In this discussion, a DMAPP application is an application that directly or indirectly uses the DMAPP library.

### 4.1 DMAPP Programming Model

Cray has supported various forms of logically shared, distributed memory (DM) programming models since the introduction of the Cray T3D. In this model, a group of processes typically run the same executable in parallel. For purposes of this discussion, such a group of related processes is termed a *job*. Although each process in a job executes in its own address space, it can access certain memory segments of other processes in the same job. This parallel model is sometimes referred to as *Single Program Multiple Data* (SPMD). DMAPP only supports SPMD style parallel jobs.

Usually the number of processes executing the application does not change over the course of a job. Processes are sometimes termed PEs (processing elements). Although each PE executes in its own address space, it can access certain memory segments of other PEs in a one-sided (PUT/GET) manner using PGAS language constructs or by invoking function calls to libraries supporting one-sided programming models (such as Cray SHMEM).
4.2 DMAPP Applications and Fork

The behavior of DM applications with respect to fork depends on which application memory segments are selected for export at link time. Fork should be avoided if the application’s stack and/or local heap are exported. The static data segment is shared between a PE and any forked children if this segment is exported. The symmetric heap is shared between PEs and any forked children.

Other DMAPP resources are also shared between a PE and any forked children. Child processes are not new PEs in the DM job. As with multi-threaded PEs, the application is responsible for setting up mutual exclusion regions around DMAPP calls.

4.3 DMAPP Applications and Threads

PEs within a DM application can be multi-threaded. However, unless the user specifies a concurrency level larger than one during DMAPP initialization, none of the functions in the DMAPP API should be considered thread-safe. Even if a concurrency level larger than one is specified, none of the functions in the DMAPP API are reentrant. For instance, they cannot be called from within a signal handler.

4.4 DMAPP Applications and File Descriptors

Handling of file descriptors in a DM application on Gemini is similar to that on Cray X2. Each PE maintains its own private file descriptors.

4.5 DMAPP Application Intra-node Communication

Data exchange between PEs on a node use the Gemini network interface. It is up to the DM model implementation to optimize for intra-node communication, if desirable.

4.6 Compiling and Launching DMAPP Applications

Source files that call DMAPP functions must include the dmapp.h header, which is included in the dmapp module. See dmapp_put.c on page 237.

Compile and link DMAPP applications on Cray XE systems using Cray-supplied compiler/linker scripts. To link a routine that makes DMAPP calls, the dmapp library is needed, which is included in the dmapp module.

By default, DMAPP mmaps the symmetric heap to huge pages and it maps the static data and private heap to base pages. It is advisable to map the static data and private heap onto huge pages. To use huge pages, link the application with libhugetlbfs. See the intro_hugepages(1) man page for more detailed information.
Use the ALPS `aprun` command to launch DMAPP applications.

### 4.7 Resiliency

DMAPP does not support error recovery in the presence of link failures. It is up to the application to deal with such error, if so desired.

### 4.8 DMAPP Remote Memory Access

Remotely accessible memory segments in a PE can be classified as either *symmetric* or *non-symmetric*.

The address of an object within a *symmetric* memory segment of a PE[X] has a known relationship to the address of this same object in the address space of another PE[Y] in the same job. Objects within these symmetric memory segments on PE[X] can be accessed in a one-sided manner by PE[Y] using address information generated locally on PE[Y].

Objects within *non-symmetric* memory segments on PE[X], can only be accessed in a one-sided manner by a second PE[Y], using address information generated by PE[X] and communicated to PE[Y].

The DMAPP implementation on the Cray XE itself does not guarantee symmetry of the symmetric heap. It is up to the DM model implementation to guarantee the symmetry of the symmetric heap or any other symmetric regions other than the statically linked data segment.

For most DM model implementations, symmetric regions are the statically linked data segment and a symmetric heap segment.

Preparing memory segments of a DM application for remote memory access is handled by DMAPP startup code. Segments which may be exported include the static data segment. The symmetric heap is always exported. At runtime, application software can determine which segments of the address space are exported using query functions. See `dmapp_get_jobinfo` on page 152.

Each exported memory segment has an associated `dmapp_seg_desc_t`. See `dmapp_seg_desc` on page 145.

You should be aware that there are trade-offs in requesting that various program segments be exported.
Since the AMD64 processor cannot be used effectively to directly load/store from exported memory on remote nodes, DMAPP provides an API for interfacing to the remote memory access (RMA) hardware mechanisms. The DMAPP RMA functions can be divided into the following categories:

- One-sided RMA functions
- RMA synchronization functions

All one-sided RMA functions (PUT type, GET type and atomic memory operations) belong to one of the following three categories:

- blocking (no suffix)
  
  The process returns from the function only after the side-effects of the remote memory access are globally visible in the system.

- non-blocking (_nb suffix)
  
  A synchronization ID (syncid) is returned to the process. The effects of the remote memory access are only assured to be globally visible in the system after the application has determined via a synchronization call (dmapp_syncid_test or dmapp_syncid_wait) whether the syncid has been retired.

- non-blocking implicit (_nbi suffix)
  
  No synchronization ID is returned to the process, the effects of the remote memory access are only assured to be globally visible in the system following a call to dmapp_gsync_test or dmapp_gsync_wait. For performance reasons, this mode is recommended for applications with many small messages, where blocking calls or using individual syncids would be expensive.

One-sided remote memory access requests require no active participation by PEs at the remote node. Remote memory segments which are targets of operations with put semantics or sources of operations with get semantics must have been exported at job startup.

The maximum number of concurrent, non-blocking requests allowable can be set by the application at DMAPP initialization. If the application attempts to initiate more non-blocking requests than this maximum, DMAPP returns an error.

There are no ordering guarantees as to completion of different non-blocking RMA requests initiated by a PE.
4.9 DMAPP API

DMAPP provides a C interface for applications. Most DMAPP functions return a status value indicating success or failure of the call. In the case of non-blocking RMA functions, this status does not indicate whether or not the remote memory access request completed successfully; it simply indicates whether the transfer request was initiated successfully. See Chapter 5, DMAPP API Reference on page 143.

4.9.1 Initialization and Query Functions

Before using any other DMAPP functions, an application must call dmapp_init or dmapp_init_ext to request and initialize resources.

New applications should use dmapp_init_ext which provides greater control over ordering modes used by RMA requests than dmapp_init provides. dmapp_init is preserved for backward compatibility.

See dmapp_init on page 151 and dmapp_init_ext on page 151.

After the last call to any other DMAPP functions, an application must call dmapp_finalize to return resources. See dmapp_finalize on page 152.

The query function dmapp_get_jobinfo returns general information about the job, such as the DMAPP version, number of PEs in the job, and symmetric heap and data segment locations. See dmapp_get_jobinfo on page 152.

A process can set RMA attributes to control the way that DMAPP handles various RMA requests. Some attributes can be set only during initialization. They will be referred to as static attributes. Others can be set multiple times over the course of the job, and will be referred to as dynamic attributes. Setting dynamic attributes does not affect RMA requests previously issued by the PE, only subsequent RMA requests. Dynamic attributes include when to switch from CPU-based mechanisms for handling RMA requests to using CPU offload mechanisms. See dmapp_set_rma_attrs on page 154, dmapp_set_rma_attrs_ext on page 155, dmapp_get_rma_attrs on page 153, and dmapp_get_rma_attrs_ext on page 153.

4.9.2 One-sided RMA Functions

RMA functions share some common arguments. For non-blocking explicit functions, the syncid argument supplies a pointer to a location in local memory which will be used by DMAPP for storing synchronization related information.

The remote address for either PUT or GET style operations is specified by a virtual address, a segment descriptor, and the remote PE.
When the virtual address is generated locally by the initiating PE, as is the case when working with symmetric data objects, the target segment descriptor supplied in the job_info structure returned by dmapp_get_job_info may be used. If the virtual address was obtained from the remote PE via some external pointer-passing mechanism, the segment descriptor from the remote PE must be used.

In addition to some common arguments, each of the data motion functions can operate on 1, 4 (DWORD), 8 (QWORD), or 16 (DQWORD) byte data types. Hardware will work most efficiently with requests that are at least DWORD aligned.

### 4.9.2.1 Contiguous Functions

The PUT functions store a contiguous block of data, starting at local memory address `source_addr`, into a contiguous block at a remote address.

For more detailed information, see [dmapp_put_nb on page 155](#), [dmapp_put_nbi on page 157](#), and [dmapp_put on page 158](#).

The GET functions load a contiguous block of data starting from a remote source address to a contiguous block starting at local memory address `target_addr`. Note that zero-length GETs are not supported.

For both PUT and GET functions, the remote address is specified by the triplet consisting of a virtual address, segment descriptor, and processor. The `nelems` parameter specifies the number of elements of type `type` to transfer. The memory region described by the remote address and `nelems` must reside in an exported memory of the remote PE.

For further information, see [dmapp_get_nb on page 159](#), [dmapp_get_nbi on page 160](#), and [dmapp_get on page 161](#).

### 4.9.2.2 Strided Functions

The strided PUT functions deliver data starting at a local memory address, using a specified stride, to a remote target address, using a separately specified stride.

The strided GET functions load data starting from a remote memory address using a specified stride and copy the data to a local memory address using a separately specified stride. For more information, see [dmapp_iget_nb on page 166](#), [dmapp_iget_nbi on page 167](#), and [dmapp_iget on page 169](#).

For both strided PUT and GET functions, the remote address is specified by the triplet consisting of a virtual address, segment descriptor, and PE. The remote memory region described by the remote address, stride, and the number of elements to transfer must reside in an exported segment of remote memory.

For more information, see [dmapp_iput_nb on page 163](#), [dmapp_iput_nbi on page 164](#), and [dmapp_iput on page 165](#).
4.9.2.3 Scatter/Gather Functions

The SCATTER functions PUT elements of a contiguous block of data in local memory to a remote memory location using multiple offset values.

The remote address for each element is specified by the triplet consisting of the virtual address, segment descriptor and target process, plus an offset value.

The memory region defined by the remote address, the largest offset in the array, and the number of elements transferred must be within an exported memory segment of the remote memory.

These functions are also referred to as indexed PUT functions and begin with the string `dmapp_ixput`. For more detail, see `dmapp_ixput_nb` on page 170, `dmapp_ixput_nbi` on page 171, and `dmapp_ixput` on page 172.

The GATHER functions GET separate elements from remote memory locations placed at various offsets, to contiguous local memory. The remote address for each element is specified by the triplet consisting of the virtual address, segment descriptor and remote process, plus an offset value.

The memory region defined by the remote address, the largest offset in the array, and the number of elements transferred must be within an exported memory segment of the remote memory.

The Indexed GET functions are `dmapp_ixget_nb` on page 173, `dmapp_ixget_nbi` on page 175, `dmapp_ixget` on page 176.

4.9.2.4 PE-strided Functions

The following functions provide PUT (broadcast), GATHER, and SCATTER to remotely accessible addresses across a set of PEs in a DMAPP job. Note that none of these are collective operations. These routines are best used when a small amount of data needs to be broadcast, scattered to, or collected from a set of PEs.

The PUT (broadcast) functions with indexed PE stride deliver data from local memory to the remote memory of multiple PEs within a DMAPP job. When the transfer is complete, each remote PE will have a copy of the contents of the original source buffer. See `dmapp_put_ixpe_nb` on page 178, `dmapp_put_ixpe` on page 181, `dmapp_put_ixpe_nbi` on page 179.

The GATHER functions with indexed PE stride gather data from the remote memory of multiple PEs within a DMAPP job, and concatenate it in local memory. The remote address ranges must be exported for each PE and the remote addresses must be symmetric. See `dmapp_gather_ixpe_nb` on page 186, `dmapp_gather_ixpe_nbi` on page 187, and `dmapp_gather_ixpe` on page 189.
The SCATTER functions with indexed PE stride deliver data starting at an address in local memory to multiple remote PEs within a DMAPP job. Each target PE receives a different portion of the local source data. See 
\texttt{dmapp\_scatter\_ixpe\_nb} on page 182, \texttt{dmapp\_scatter\_ixpe\_nbi} on page 183, and \texttt{dmapp\_scatter\_ixpe} on page 185.

For all PE-strided functions, the remote address must be a symmetric address; it must lie in the statically linked data segment or the symmetric heap.

### 4.9.2.5 DMAPP AMO Functions

In addition to PUT and GET RMA functionality, DMAPP also provides support for using atomic memory operation (AMO) RMA requests. The set of AMO functions are modeled on the set provided on the Cray X2 systems. AMOs can be used both for synchronization and at-memory style operations. AMOs are restricted to operating on 8-byte (also referred to as \textit{qword}, \textit{qw}, or \textit{quad word}) data types, located in a remote PE.

As with RMA functions, the remote memory location must reside in an exported memory segment of remote PE.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Data Returned in Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFADD</td>
<td>Atomic Fetch and ADD</td>
<td>yes</td>
</tr>
<tr>
<td>AFAX</td>
<td>Atomic Fetch and XOR</td>
<td>yes</td>
</tr>
<tr>
<td>ACSWAP</td>
<td>Compare and swap</td>
<td>yes</td>
</tr>
<tr>
<td>AADD</td>
<td>Atomic ADD</td>
<td>no</td>
</tr>
<tr>
<td>AFAND</td>
<td>Atomic fetch and AND</td>
<td>yes</td>
</tr>
<tr>
<td>AFOR</td>
<td>Atomic fetch and OR</td>
<td>yes</td>
</tr>
<tr>
<td>AFXOR</td>
<td>Atomic fetch and XOR</td>
<td>yes</td>
</tr>
<tr>
<td>AAND</td>
<td>Atomic AND</td>
<td>no</td>
</tr>
<tr>
<td>AOR</td>
<td>Atomic OR</td>
<td>no</td>
</tr>
<tr>
<td>AXOR</td>
<td>Atomic XOR</td>
<td>no</td>
</tr>
</tbody>
</table>

The scalar-type blocking and non-blocking Atomic Memory Operations (AMO) functions where no result is returned are named \texttt{dmapp\_op\_qw}, \texttt{dmapp\_op\_qw\_nb}, \texttt{dmapp\_op\_qw\_nbi}, where \textit{op} is either AADD, AAND, AOR, or AXOR.

For more detail on all AMO functions, see \texttt{dmapp\_aadd\_qw\_nb} on page 190 through \texttt{dmapp\_acswap\_qw} on page 220.
The scalar-type blocking and non-blocking Atomic Memory Operations (AMO) functions in which the result is returned in the response are named dmapp_op_qw, dmapp_op_qw_nb, dmapp_op_qw_nbi, where op is either ACSWAP, AFADD, AFAND, AFAX, AFOR, or AFXOR.

### 4.9.2.6 DMAPP Synchronization Functions

DMAPP applications use synchronization functions to determine when locally initiated, non-blocking RMA requests have completed.

A process can determine when the effects of a non-blocking explicit RMA function call are globally visible in the system by using dmapp_sync_test() or dmapp_sync_wait(), both of which return information about the specified syncid.

`dmapp_sync_test()` immediately returns a value indicating whether all RMA requests associated with syncid have completed. `dmapp_sync_wait()` only returns after all RMA requests associated with syncid have completed. See `dmapp_syncid_test` on page 221, and `dmapp_syncid_wait` on page 222.

A process can determine when the side effects of one or more non-blocking implicit RMA function calls are globally visible in the system by using `dmapp_gsync_test()` or `dmapp_gsync_wait()` functions, both of which refer to all remote memory accesses associated with previously issued non-blocking implicit RMA requests; therefore, a syncid is not relevant.

`dmapp_gsync_test()` immediately returns with a value indicating whether all remote memory accesses associated with previously issued non-blocking implicit RMA function calls are globally visible in the system. `dmapp_gsync_wait()` only returns after all non-blocking implicit RMA requests are globally visible in the system. See `dmapp_gsync_test` on page 222 and `dmapp_gsync_wait` on page 223.

### 4.9.2.7 Collective Operations

DMAPP provides a set of routines to perform global reduction operations across multiple processes. These collective operations are intended to be used for small quantities of data such as reductions on 4-7 double precision numbers, broadcast of up to 64 bytes, etc.

Processes associated with each other, in the context of collective operations, are referred to as a process set, or pset.

Prior to pset creation, the DMAPP application must first initialize a dmapp_c_pset_desc_t structure to describe the processes contained in the pset. See `dmapp_c_pset_desc_t` on page 150.
A pset is created using a two step process. First, each rank which will be participating in DMAPP collective operations locally defines the ranks involved in the pset using the `dmapp_c_pset_create` function. The ranks then use an out-of-band synchronization mechanism to insure that all ranks contained in the pset have completed the call to `dmapp_c_pset_create`, at which point the pset can be exported by each rank using the `dmapp_c_pset_export` function. Rank information can be obtained from the `dmapp_get_job_info` function, or from the PMI `PMI_Get_rank_in_app` function. See `dmapp_c_pset_create` on page 223 and `dmapp_c_pset_export` on page 224.

To initiate a global reduction operation over a previously exported process set, an application will use the `dmapp_c_greduce_start()` function. The `dmapp_c_barrier_join()` function initiates a barrier join operation on a previously exported pset.

The interfaces are non-blocking, with the exception of `dmapp_c_pset_wait()` which instructs an application to wait for completion of a collective operation for a given pset. Note that only one outstanding, non-blocking collective operation is allowed on a given pset.

An application may use `dmapp_c_pset_cancel_op()` to cancel an outstanding collective operation on a pset.

See `dmapp_c_greduce_start` on page 227, `dmapp_c_barrier_join` on page 225, `dmapp_c_pset_wait` on page 230, and `dmapp_c_pset_cancel_op` on page 226.

The DMAPP application must use `dmapp_c_pset_destroy()` to free internal resources associated with the pset. See `dmapp_c_pset_destroy` on page 226.

**Note:** DMAPP makes no distinction between ranks in a pset which are co-located on the same node and those that are not. It is very likely that upper-level software can more efficiently handle on-node components of, for example, barrier operations than DMAPP can. It is expected that for best performance, such software will use DMAPP only for inter-node components of collective operations, and user lower-latency, shared memory methods for intra-node stages of these operations.
4.9.3 Symmetric Heap Functions

DMAPP provides routines for allocating and releasing symmetric heap memory. The DMAPP application is responsible for preserving symmetry of this heap memory. This is achieved by ensuring that all PEs in a job make the same calls to the symmetric heap management functions in the same sequence, involving the same amount of memory. The DMAPP application controls the size of the symmetric heap at startup.

```c
void *dmapp_sheap_malloc(IN size_t size)
```

This function allocates `size` bytes memory from the symmetric heap. Equality of addresses across PEs is not guaranteed.

```c
void *dmapp_sheap_realloc(IN void *ptr, IN size_t size)
```

This function changes the size of the block to which `ptr` points to the `size` (in bytes) specified by `size`. Equality of addresses across PEs is not guaranteed.

```c
void dmapp_sheap_free(IN void *ptr)
```

This function frees a block of memory previously allocated by `dmapp_sheap_malloc` or `dmapp_sheap_realloc`.

4.9.4 Checkpoint Restart Functions

These functions allow a BLCR callback function to prepare an application for checkpointing and to perform the checkpoint/restart operation. See `dmapp_checkpoint` on page 235 and `dmapp_restart` on page 235.
Using the GNI and DMAPP APIs
This chapter contains reference information for enumerations, structures, and functions contained in the DMAPP API. Your application must include the dmapp.h file when using this API.

5.1 DMAPP Enumerations

5.1.1 dmapp_type

The dmapp_type_t enumeration defines the valid types supplied by the type input parameter to all data motion functions.

5.1.1.1 Synopsis

typedef enum dmapp_type {
    DMAPP_DQW = 0,
    DMAPP_QW,
    DMAPP_DW,
    DMAPP_BYTE
} dmapp_type_t;

5.1.1.2 Constants

DMAPP_DQW Indicates a double quad (16 byte) word.

DMAPP_QW Indicates a quad (8 byte) word.

DMAPP_DW Indicates a double (4 byte) word.

DMAPP_BYTE Indicates a byte. This option does not provide good performance.

5.1.2 dmapp_routing_type

The dmapp_routing_type_t enumeration defines the valid routing modes to be supplied to the relaxed_ordering fields of the RMA attributes structures dmapp_rma_attrs on page 146 and dmapp_rma_attrs_ext on page 148.

typedef enum uint8_t {
    DMAPP_ROUTING_IN_ORDER = 0, /* hash off, adapt off */
    DMAPP_ROUTING_DETERMINISTIC, /* hash on, adapt off */
    DMAPP_ROUTING_ADAPTIVE /* hash off, adapt on */
} dmapp_routing_type_t;
5.1.3 dmapp_pi_reg_type

The dmapp_pi_reg_type_t enumeration defines the modes of PI access ordering to be used by DMAPP during memory registration with uGNI; therefore, these modes apply to the data and symmetric heap and any user or dynamically mapped regions. See PI_ordering field of the extended RMA attributes structure dmapp_rma_attrs_ext on page 148.

These modes do not affect GET operations.

Strict ordering ensures that posted and non-posted writes arrive at the target in strict order. Default and relaxed ordering impose no ordering constraints, therefore if an application requires the global visibility of data (for example, after a blocking put or gsync/fence), it must perform extra synchronization in the form of a remote GET from the target node in order to ensure that written data is globally visible.

typedef enum dmapp_pi_reg_type {
    DMAPP_PI_ORDERING_STRICT = 0, /* Strict PI (P_PASS_PW=0, NP_PASS_PW=0) */
    DMAPP_PI_ORDERING_DEFAULT, /* Default GNI PI (P_PASS_PW=0, NP_PASS_PW=1) */
    DMAPP_PI_ORDERING_RELAXED /* Relaxed PI ordering (P_PASS_PW=1, NP_PASS_PW=1) */
} dmapp_pi_reg_type_t;

5.1.4 dmapp_c_op

The dmapp_c_op enumeration defines the valid op input parameters required by dmapp_c_greduce_start().

typedef enum dmapp_c_op {
    DMAPP_C_SUM = 201,
    DMAPP_C_MAX,
    DMAPP_C_MIN,
    DMAPP_C_PROD,
    DMAPP_C_BAND,
    DMAPP_C_LAND,
    DMAPP_C_BAND,
    DMAPP_C_LOR,
    DMAPP_C_BOR,
    DMAPP_C_LXOR,
    DMAPP_C_BXOR,
    DMAPP_C_MINLOC,
    DMAPP_C_MAXLOC,
    DMAPP_C_OP_LAST
} dmapp_c_op_t;

5.1.5 dmapp_c_type

The dmapp_c_type_t enumeration defines the valid types required by the type input parameter to dmapp_c_greduce_start() and dmapp_c_greduce_nelems_max().
typedef enum dmapp_c_type {
    DMAPP_C_INT32 = 101,
    DMAPP_C_UINT32,
    DMAPP_C_INT64,
    DMAPP_C_UINT64,
    DMAPP_C_FLOAT,
    DMAPP_C_DOUBLE,
    DMAPP_C_COMPLEX8,
    DMAPP_C_COMPLEX16,
    DMAPP_C_FLOAT_UINT64,
    DMAPP_C_DOUBLE_UINT64,
    DMAPP_C_INT32_UINT64,
    DMAPP_C_INT64_UINT64,
    DMAPP_C_TYPE_LAST
} dmapp_c_type_t;

5.2 DMAPP Structures

5.2.1 dmapp_seg_desc

The dmapp_seg_desc structure is a memory segment descriptor, with an address and length.

5.2.1.1 Synopsis

typedef struct dmapp_seg_desc {
    void   *addr;
    size_t  len;
    gni_mem_handle_t memhndl;
    uint16_t flags;
    void   *reserved;
} dmapp_seg_desc_t;

5.2.1.2 Members

addr A pointer to the address for the memory segment.
len The currently mapped size of the segment, in bytes.
memhndl Memory handle for the region; automatically obtained at initialization or obtained from a previous call to MemRegister.
flags For internal use only.
reserved For internal use only.

5.2.2 dmapp_jobinfo

The dmapp_jobinfo structure contains general information relevant to the job.
5.2.2.1 Synopsis

typedef struct dmapp_jobinfo {
    int version;
    int hw_version;
    int npes;
    dmapp_pe_t pe;
    dmapp_seg_desc_t data_seg;
    dmapp_seg_desc_t sheap_seg;
} dmapp_jobinfo_t;

5.2.2.2 Members

version The version of DMAPP that this job uses.

hw_version The hardware version of the system. The current version is
DMAPP_GNI_HW_MAJOR_GEMINI.

npes The number of processing elements in use for the entire job.

pe The processing element number, in [0, npes-1].

data_seg The data segment in memory that this job is using.

sheap_seg The symmetric heap memory that this job is using.

5.2.3 dmapp_rma_attrs

The dmapp_rma_attrs structure sets RMA attributes to control the way in
which DMAPP handles various RMA requests. Some attributes can be set during
initialization only, others can be set multiple times over the course of a job.

5.2.3.1 Synopsis

typedef struct dmapp_rma_attrs {
    uint32_t max_outstanding_nb;
    uint32_t offload_threshold;
    uint8_t put_relaxed_ordering;
    uint8_t get_relaxed_ordering;
    uint8_t max_concurrency;
} dmapp_rma_attrs_t;

5.2.3.2 Members

max_outstanding_nb

The maximum number of outstanding non-blocking requests
supported. You can only specify this flag during initialization. The
following is the range of valid values to be supplied:

[DMAPP_MIN_OUTSTANDING_NB, .., DMAPP_MAX_OUTSTANDING_NB]
Setting the value to one of the extremes may lead to a slowdown. The recommended value is DMAPP_DEF_OUTSTANDING_NB. Users can experiment with the value to find the optimal setting for their application.

**offload_threshold**

The threshold, in bytes, for switching between CPU-based mechanisms and CPU offload mechanisms. This value can be specified at any time and can use any value. The default setting is DMAPP_OFFLOAD_THRESHOLD. Very small or very large settings may lead to suboptimal performance. The default value is 4k bytes. Consider how to best set this threshold. While a threshold increase may increase CPU availability, it may also increase transfer latency due to BTE involvement.

**put_relaxed_ordering**

Specifies the type of routing to be used. Applies to RMA requests with PUT semantics and all AMOs. The default is DMAPP_ROUTING_DETERMINISTIC. The value can be specified at any time. Note that DMAPP_ROUTING_IN_ORDER may result in poor performance. Valid settings are:

- DMAPP_ROUTING_IN_ORDER
- DMAPP_ROUTING_DETERMINISTIC
- DMAPP_ROUTING_ADAPTIVE

**get_relaxed_ordering**

Specifies the type of routing to be used. Applies to RMA requests with GET semantics. The default is DMAPP_ROUTING_ADAPTIVE. The value can be specified at any time. Note that DMAPP_ROUTING_IN_ORDER may result in poor performance. Valid settings are:

- DMAPP_ROUTING_IN_ORDER
- DMAPP_ROUTING_DETERMINISTIC
- DMAPP_ROUTING_ADAPTIVE

**max_concurrency**

The maximum number of threads that can access DMAPP. You can only use this when thread-safety is enabled. The default is 1. You can only specify this during initialization and it must be $\geq 1$. 


5.2.4 dmapp_rma_attrs_ext

The dmapp_rma_attrs_ext structure sets extended RMA attributes to control the way in which DMAPP handles various RMA requests. Some attributes can be set during initialization only, others can be set multiple times over the course of a job.

5.2.4.1 Synopsis

typedef struct dmapp_rma_attrs_ext {
    uint32_t max_outstanding_nb;
    uint32_t offload_threshold;
    uint8_t put_relaxed_ordering;
    uint8_t get_relaxed_ordering;
    uint8_t max_concurrency;
    uint8_t PI_ordering;
    uint8_t unused[32];
} dmapp_rma_attrs_t;

5.2.4.2 Members

max_outstanding_nb

The maximum number of outstanding non-blocking requests supported. You can only specify this flag during initialization. The following is the range of valid values to be supplied:

[DMAPP_MIN_OUTSTANDING_NB, ..., DMAPP_MAX_OUTSTANDING_NB]

Setting the value to one of the extremes may lead to a slowdown. The recommended value is DMAPP_DEF_OUTSTANDING_NB. Users can experiment with the value to find the optimal setting for their application.

offload_threshold

The threshold, in bytes, for switching between CPU-based mechanisms and CPU offload mechanisms. This value can be specified at any time and can use any value. The default setting is DMAPP_OFFLOAD_THRESHOLD. Very small or very large settings may lead to suboptimal performance. The default value is 4k bytes. Consider how to best set this threshold. While a threshold increase may increase CPU availability, it may also increase transfer latency due to BTE involvement.
put_relaxed_ordering

Specifies the type of routing to be used. See dmapp_routing_type on page 143. Applies to RMA requests with PUT semantics and all AMOs. The default is DMAPP_ROUTING_DETERMINISTIC. The value can be specified at any time. Note that DMAPP_ROUTING_IN_ORDER guarantees the requests arrive in order and may result in poor performance. Valid settings are:

- DMAPP_ROUTING_IN_ORDER
- DMAPP_ROUTING_DETERMINISTIC
- DMAPP_ROUTING_ADAPTIVE

get_relaxed_ordering

Specifies the type of routing to be used. Applies to RMA requests with GET semantics. The default is DMAPP_ROUTING_ADAPTIVE. The value can be specified at any time. Note that DMAPP_ROUTING_IN_ORDER may result in poor performance. Valid settings are:

- DMAPP_ROUTING_IN_ORDER
- DMAPP_ROUTING_DETERMINISTIC
- DMAPP_ROUTING_ADAPTIVE

max_concurrency

The maximum number of threads that can access DMAPP. You can only use this when thread-safety is enabled. The default is 1. You can only specify this during initialization and it must be >= 1.

PI_ordering

Defines the PI ordering registration flags used by DMAPP when registering all memory regions with GNI. Applies to the data, symmetric heap, and user or dynamically mapped regions. Possible values are defined by dmapp_pi_reg_type on page 144. The default is DMAPP_PI_RELAXED_ORDERING.

5.2.5 dmapp_syncid

The dmapp_syncid structure contains a pointer to the synchronization ID that is used by a non-blocking explicit RMA function.

5.2.5.1 Synopsis

typedef struct dmapp_syncid *dmapp_syncid_handle_t;
5.2.6 dmapp_c_pset_desc_t

The DMAPP application initializes a dmapp_c_pset_desc structure prior to pset creation.

A symmetric heap address and length (returned from dmapp_sheap_malloc()) must be supplied in the concat_buf and concat_buf_size fields. Generally the larger the concat_buf supplied at pset creation, the better the performance of the concatenate function, especially as the number of ranks increases.

5.2.6.1 Synopsis

```c
typedef struct {
    uint32_t n_pes;
    uint32_t *vec_pes;
} dmapp_c_pset_delimiter_vec_t;

typedef struct {
    uint32_t n_pes;
    uint32_t base_pe;
    uint32_t stride_pe;
} dmapp_c_pset_delimiter_strided_t;

typedef struct {
    void *concat_buf;
    uint64_t concat_buf_size;
    dmapp_c_pset_delimiter_type_t type;
    union {
        dmapp_c_pset_delimiter_vec_t vec_type;
        dmapp_c_pset_delimiter_strided_t stride_type;
    } u;
} dmapp_c_pset_desc_t;
```

5.2.7 dmapp_c_pset_handle_t

5.2.7.1 Synopsis

```c
typedef struct dmapp_c_pset *dmapp_c_pset_handle_t;
```
5.3 DMAPP Functions

5.3.1 dmapp_init

The dmapp_init function initializes resources for a DMAPP job. All DMAPP applications must call this function before using other DMAPP functions. In a threaded application, this function should only be called once.

After the last call to any other DMAPP functions, an application must call a finalization function: dmapp_return_t dmapp_finalize(void).

5.3.1.1 Synopsis

dmapp_return_t dmapp_init(
    IN  dmapp_rma_attrs_t *requested_attrs,
    OUT dmapp_rma_attrs_t *actual_attrs);

5.3.1.2 Parameters

requested_attrs

Pointer to the desired job attributes. See dmapp_rma_attrs on page 146.

actual_attrs

The actual job attributes.

5.3.1.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more parameters are invalid.

DMAPP_RC_RESOURCE_ERROR

An error occurred during initialization.

5.3.2 dmapp_init_ext

The dmapp_init_ext function initializes resources for a DMAPP job. All DMAPP applications must call this function before using other DMAPP functions. In a threaded application, this function should only be called once.

After the last call to any other DMAPP functions, an application must call a finalization function: dmapp_return_t dmapp_finalize(void).
5.3.2.1 Synopsis

```c
dmapp_return_t dmapp_init_ext(
    IN dmapp_rma_attrs_ext_t *requested_attrs,
    OUT dmapp_rma_attrs_ext_t *actual_attrs);
```

5.3.2.2 Parameters

`requested_attrs`

Pointer to the desired job attributes. See `dmapp_rma_attrs_ext` on page 148.

`actual_attrs`

The actual job attributes.

5.3.2.3 Return Codes

- `DMAPP_RC_SUCCESS`
  
  The operation completed successfully.

- `DMAPP_RC_INVALID_PARAM`
  
  One or more parameters are invalid.

- `DMAPP_RC_RESOURCE_ERROR`
  
  An error occurred during initialization.

5.3.3 `dmapp_finalize`

The `dmapp_finalize` function synchronizes and cleans up DMAPP resources.

All DMAPP applications must call this function when it has finished using all DMAPP functions.

5.3.3.1 Synopsis

```c
dmapp_return_t dmapp_finalize(void);
```

5.3.3.2 Return Codes

- `DMAPP_RC_SUCCESS`
  
  The operation completed successfully.

5.3.4 `dmapp_get_jobinfo`

The `dmapp_get_jobinfo` function returns a pointer to the data structure `dmapp_jobinfo` on page 145 which contains general information about the job.
5.3.4.1 Synopsis

```c
dmapp_return_t dmapp_get_jobinfo(
    OUT dmapp_jobinfo_t *info);
```

5.3.4.2 Parameters

- `info` Returns a pointer to the current information about the job.

5.3.4.3 Return Codes

- **DMAPP_RC_SUCCESS**
  - The operation completed successfully.
- **DMAPP_RC_INVALID_PARAM**
  - The input parameter is invalid.

5.3.5 `dmapp_get_rma_attrs`

The `dmapp_get_rma_attrs` function returns RMA attributes of a DMAPP thread.

5.3.5.1 Synopsis

```c
dmapp_return_t dmapp_get_rma_attrs(
    OUT dmapp_rma_attrs_t *attrs);
```

5.3.5.2 Parameters

- `attrs` Current RMA attributes of the thread in local storage.

5.3.5.3 Return Codes

- **DMAPP_RC_SUCCESS**
  - The operation completed successfully.
- **DMAPP_RC_INVALID_PARAM**
  - Input parameter is invalid.

5.3.6 `dmapp_get_rma_attrs_ext`

The `dmapp_get_rma_attrs_ext` function returns extended RMA attributes of a DMAPP thread.
In order to facilitate forward compatibility using the extended capabilities in the extended RMA attributes structures, the user should call dmapp_get_rma_attrs_ext() in order to sample the default RMA attributes and then modify the parameters they wish to change, before calling dmapp_init_ext() or dmapp_set_rma-attrs_ext().

5.3.6.1 Synopsis

```c
dmapp_return_t dmapp_get_rma_attrs_ext(
    OUT dmapp_rma_attrs_ext_t *attrs);
```

5.3.6.2 Parameters

`attrs`  
Current extended RMA attributes of the thread in local storage.

5.3.6.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.
- **DMAPP_RC_INVALID_PARAM**
  Input parameter is invalid.

5.3.7 dmapp_set_rma_attrs

The `dmapp_set_rma_attrs` function sets dynamic RMA attributes for a DMAPP thread using local storage. This allows each thread to independently set the RMA attributes.

5.3.7.1 Synopsis

```c
dmapp_return_t dmapp_set_rma_attrs(
    IN  dmapp_rma_attrs_t *requested_attr;
    OUT dmapp_rma_attrs_t *actual_attr);`
```

5.3.7.2 Parameters

- `requested_attr`
  Pointer to desired job attributes.

- `actual_attr`
  Returns a pointer to the actual job attributes.
5.3.7.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

5.3.8 dmapp_set_rma_attrs_ext

The dmapp_set_rma_attrs_ext function sets dynamic, extended RMA attributes for a DMAPP thread using local storage. This allows each thread to independently set the extended RMA attributes.

5.3.8.1 Synopsis

```
dmapp_return_t dmapp_set_rma_attrs_ext(
   IN dmapp_rma_attrs_ext_t *requested_attrs,
   OUT dmapp_rma_attrs_ext_t *actual_attrs);
```

5.3.8.2 Parameters

requested_attrs

Pointer to desired job attributes.

actual_attrs

Returns a pointer to the actual job attributes.

5.3.8.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

5.3.9 dmapp_put_nb

The dmapp_put_nb function is a non-blocking explicit PUT. dmapp_put_nb stores a contiguous block of data, starting at the address indicated by source_addr, from local memory into a contiguous block at a remote address. The remote address is specified by the triplet virtual address target_addr, segment descriptor target_seg, and the target process target_pe. nelems specifies the number of elements of type type to be transferred. The memory region defined by target_addr and nelems must be within an exported memory segment of target_pe.
5.3.9.1 Synopsis

```c
dmapp_return_t dmapp_put_nb(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN void *source_addr,
    IN uint64_t nelems,
    IN dmapp_type_t type,
    OUT dmapp_syncid_handle_t *syncid);
```

5.3.9.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer.

- **target_seg**
  Pointer to the segment descriptor of the target buffer.

- **target_pe**
  The target processing element.

- **source_addr**
  Pointer to the address of the source buffer.

- **nelems**
  Number of elements to transfer.

- **type**
  The type of elements to transfer.

- **syncid**
  Pointer to the synchronization ID.

5.3.9.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_RESOURCE_ERROR**
  A resource error occurred.
The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.10 dmapp_put_nbi

The dmapp_put_nbi function is a non-blocking implicit PUT. dmapp_put_nbi stores a contiguous block of data, starting at the address indicated by source_addr, from local memory into a contiguous block at a remote address. The remote address is specified by the triplet virtual address target_addr, segment descriptor target_seg, and the target process target_pe. nelems specifies the number of elements of type type to be transferred. The memory region defined by target_addr and nelems must be within an exported memory segment of target_pe.

5.3.10.1 Synopsis

```c
dmapp_return_t dmapp_put_nbi(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN void *source_addr,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

5.3.10.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target_addr</td>
<td>Pointer to the address of a target buffer.</td>
</tr>
<tr>
<td>target_seg</td>
<td>Pointer to a segment descriptor of a target buffer.</td>
</tr>
<tr>
<td>target_pe</td>
<td>The target processing element.</td>
</tr>
<tr>
<td>source_addr</td>
<td>Address of the source buffer.</td>
</tr>
<tr>
<td>nelems</td>
<td>The number of elements to transfer.</td>
</tr>
<tr>
<td>type</td>
<td>The type of elements to transfer.</td>
</tr>
</tbody>
</table>
5.3.10.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RCRESOURCE_ERROR

A resource error occurred.

5.3.11 dmapp_put

The dmapp_put function is a blocking PUT. dmapp_put stores a contiguous block of data, starting at the address indicated by source_addr, from local memory into a contiguous block at a remote address. The remote address is specified by the triplet virtual address target_addr, segment descriptor target_seg, and the target process target_pe. nelems specifies the number of elements of type type to be transferred. The memory region defined by target_addr and nelems must be within an exported memory segment of target_pe.

5.3.11.1 Synopsis

```c
dmapp_return_t dmapp_put(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN void *source_addr,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

5.3.11.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer.

- **target_seg**
  Pointer to the segment descriptor of the target buffer.

- **target_pe**
  The target processing element.

- **source_addr**
  Pointer to the address of the source buffer.

- **nelems**
  The number of elements to transfer.

- **type**
  The type of elements to transfer.
5.3.11.3 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_INVALID_PARAM
One or more input parameters is invalid.

DMAPP_RC_RESOURCE_ERROR
A resource error occurred.

DMAPP_RC_NO_SPACE
The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

DMAPP_RC_TRANSACTION_ERROR
A transaction error has occurred.

5.3.12 dmapp_get_nb

The dmapp_get_nb function is a non-blocking explicit GET. dmapp_get_nb loads from a contiguous block of data starting from a remote source address and returning the data into a contiguous block starting at address target_addr in local memory. The remote address is specified by the triplet virtual address source_addr, segment descriptor source_seg and source process source_pe. The nelems parameter specifies the number of elements of type type to transfer. The memory region described by the remote address and nelems must reside in an exported memory of source_pe. Note that zero-length GETs are not supported.

5.3.12.1 Synopsis

dmapp_return_t dmapp_get_nb(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN uint64_t nelems,
    IN dmapp_type_t type,
    OUT dmapp_syncid_handle_t *syncid);
5.3.12.2 Parameters

\[ target\_addr \]

Pointer to the address of the target buffer.

\[ source\_addr \]

Pointer to the address of the source buffer.

\[ source\_seg \]

Pointer to a segment descriptor of a source buffer.

\[ source\_pe \]

The source processing element.

\[ nelems \]

The number of elements to transfer.

\[ type \]

The type of elements to transfer.

\[ syncid \]

Pointer to a synchronization ID.

5.3.12.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_RESOURCE_ERROR

A resource error occurred.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.

5.3.13 dmapp_get_nbi

The `dmapp_get_nbi` function is a non-blocking implicit GET. `dmapp_get_nbi` loads from a contiguous block of data starting from a remote source address and returning the data into a contiguous block starting at address `target_addr` in local memory. The remote address is specified by the triplet virtual address `source_addr`, segment descriptor `source_seg` and source process `source_pe`. The `nelems` parameter specifies the number of elements of type `type` to transfer. The memory region described by the remote address and `nelems` must reside in an exported memory of `source_pe`. Note that zero-length GETs are not supported.
5.3.13.1 Synopsis

```c
dmapp_return_t dmapp_get_nbi(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

5.3.13.2 Parameters

- `target_addr`: Pointer to the address of the target buffer.
- `source_addr`: Pointer to the address of the source buffer.
- `source_seg`: Pointer to the segment descriptor of the source buffer.
- `source_pe`: The source processing element.
- `nelems`: The number of elements to transfer.
- `type`: The type of elements to transfer.

5.3.13.3 Return Codes

- `DMAPP_RC_SUCCESS`: The operation completed successfully.
- `DMAPP_RC_INVALID_PARAM`: One or more input parameters is invalid.
- `DMAPP_RC_RESOURCE_ERROR`: A resource error occurred.

5.3.14 `dmapp_get`

The `dmapp_get` function is a blocking GET. `dmapp_get` loads from a contiguous block of data starting from a remote source address and returning the data into a contiguous block starting at address `target_addr` in local memory. The remote address is specified by the triplet virtual address `source_addr`, segment descriptor `source_seg` and source process `source_pe`. The `nelems` parameter specifies the number of elements of type `type` to transfer. The memory region described by the remote address and `nelems` must reside in an exported memory of `source_pe`. Note that zero-length GETs are not supported.
5.3.14.1 Synopsis

```c
dmapp_return_t dmapp_get_nbi(
  IN void * target_addr,
  IN void * source_addr,
  IN dmapp_seg_desc_t *source_seg,
  IN dmapp_pe_t source_pe,
  IN uint64_t nelems,
  IN dmapp_type_t type);
```

5.3.14.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer.

- **source_addr**
  Pointer to the address of the source buffer.

- **source_seg**
  Pointer to the segment descriptor of the source buffer.

- **source_pe**
  The source processing element.

- **nelems**
  The number of elements to transfer.

- **type**
  The type of elements to transfer.

5.3.14.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_RESOURCE_ERROR**
  A resource error occurred.

- **DMAPP_RC_NO_SPACE**
  The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.

- **DMAPP_RC_TRANSACTION_ERROR**
  A transaction error has occurred.
5.3.15 dmapp_iput_nb

The dmapp_iput_nb function is a non-blocking explicit strided PUT.

5.3.15.1 Synopsis

dmapp_return_t dmapp_iput_nb(
    IN void * target_addr,
    IN dmapp_seg_desc_t * target_seg,
    IN dmapp_pe_t target_pe,
    IN void * source_addr,
    IN ptrdiff_t tst,
    IN ptrdiff_t sst,
    IN uint64_t nelems,
    IN dmapp_type_t type,
    OUT dmapp_syncid_handle_t * syncid);

5.3.15.2 Parameters

    target_addr
        Pointer to the address of the target buffer.

    target_seg
        Pointer to the segment descriptor of a target buffer.

    target_pe
        Target processing element.

    source_addr
        Pointer to the address of the source buffer.

    tst
        Target stride (>= 1).

    sst
        Source stride (>= 1).

    nelems
        Number of elements to transfer.

    type
        Type of elements to transfer.

    syncid
        Returns the synchronization ID.

5.3.15.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.
DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.16 dmapp_iput_nbi

The dmapp_iput_nbi function is a non-blocking implicit strided PUT.

5.3.16.1 Synopsis

```c
dmapp_return_t dmapp_iput_nbi(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN void *source_addr,
    IN ptrdiff_t tst,
    IN ptrdiff_t sst,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

5.3.16.2 Parameters

- `target_addr`: Pointer to the address of the target buffer.
- `target_seg`: Pointer to the segment descriptor of the target buffer.
- `target_pe`: Target processing element.
- `source_addr`: Pointer to the address of the source buffer.
- `tst`: Target stride.
- `sst`: Source stride.
- `nelems`: Number of elements to transfer.
- `type`: Type of elements to transfer.
5.3.16.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.17 dmapp_iput

The dmapp_iput function is a blocking strided PUT.

5.3.17.1 Synopsis

    dmapp_return_t dmapp_iput(
        IN void *target_addr,
        IN dmapp_seg_desc_t *target_seg,
        IN dmapp_pe_t target_pe,
        IN void *source_addr,
        IN ptrdiff_t tst,
        IN ptrdiff_t sst,
        IN uint64_t nelems,
        IN dmapp_type_t type);

5.3.17.2 Parameters

    target_addr
        Pointer to the address of the target buffer.

    target_seg
        Pointer to the segment descriptor of the target buffer.

    target_pe
        Target processing element.

    source_addr
        Pointer to the address of the source buffer.

    tst
        Target stride.

    sst
        Source stride.

    nelems
        Number of elements to transfer.

    type
        Type of elements to transfer.
5.3.17.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

DMAPP_RC_TRANSACTION_ERROR

A transaction error has occurred.

5.3.18 dmapp_iget_nb

The dmapp_iget_nb function is a non-blocking explicit strided GET.

5.3.18.1 Synopsis

```c
dmapp_return_t dmapp_iget_nb(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN ptrdiff_t tst,
    IN ptrdiff_t sst,
    IN uint64_t nelems,
    IN dmapp_type_t type,
    OUT dmapp_syncid_handle_t *syncid);
```
5.3.18.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer.

- **source_addr**
  Pointer to the address of the source buffer.

- **source_seg**
  Pointer to the segment descriptor of the source buffer.

- **source_pe**
  Source processing element.

- **tst**
  Target stride.

- **sst**
  Source stride.

- **nelems**
  Number of elements to transfer.

- **type**
  Type of elements to transfer.

- **syncid**
  Pointer to the synchronization ID.

5.3.18.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_ALIGNMENT_ERROR**
  The source or target buffer or length is not properly Dword (4 byte) aligned.

- **DMAPP_RC_RESOURCE_ERROR**
  A resource error occurred.

- **DMAPP_RC_NO_SPACE**
  The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for **max_outstanding_nb** in the job attributes.

5.3.19 *dmapp_iget_nbi*

The *dmapp_iget_nbi* function is a non-blocking implicit strided GET.
5.3.19.1 Synopsis

```c
dmapp_return_t dmapp_iget_nbi(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN ptrdiff_t tst,
    IN ptrdiff_t sst,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

5.3.19.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer.

- **source_addr**
  Pointer to the address of the source buffer.

- **source_seg**
  Pointer to the segment descriptor of the source buffer.

- **source_pe**
  Source processing element.

- **tst**
  Target stride.

- **sst**
  Source stride.

- **nelems**
  Number of elements to transfer.

- **type**
  Type of elements to transfer.

5.3.19.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_ALIGNMENT_ERROR**
  The source or target buffer or length is not properly Dword (4 byte) aligned.

- **DMAPP_RC_RESOURCE_ERROR**
  A resource error occurred.
The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.20 dmapp_iget

The dmapp_iget function is a blocking strided GET.

5.3.20.1 Synopsis

```c
dmapp_return_t dmapp_iget(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN ptrdiff_t tst,
    IN ptrdiff_t sst,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

5.3.20.2 Parameters

- **target_addr**
  
  Pointer to the address of the target buffer.

- **source_addr**
  
  Pointer to the address of the source buffer.

- **source_seg**
  
  Pointer to the segment descriptor of the source buffer.

- **source_pe**
  
  Source processing element.

- **tst**
  
  Target stride (\geq 1).

- **sst**
  
  Source stride (\geq 1).

- **nelems**
  
  Number of elements to transfer.

- **type**
  
  Type of elements to transfer.
5.3.20.3 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_INVALID_PARAM
One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR
Source or target buffer or length not properly Dword (4 byte) aligned.

DMAPP_RC_RESOURCE_ERROR
A resource error occurred.

DMAPP_RC_NO_SPACE
The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

DMAPP_RC_TRANSACTION_ERROR
A transaction error has occurred.

5.3.21 dmapp_ixput_nb

The dmapp_ixput_nb function is a non-blocking explicit Indexed PUT.

PUT contiguous data starting at address source_addr in local memory to a remote target address using offsets specified by the tidx array. The remote address is specified by the triplet virtual address target_addr, segment descriptor target_seg and target process target_pe. nelems specifies the number of elements of type type to be transferred. Offsets in the tidx array are in units of type. The memory region described by the remote address, tidx and nelems must reside in an exported memory of target_pe.

5.3.21.1 Synopsis

dmapp_return_t dmapp_ixput_nb(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN void *source_addr,
    IN ptrdiff_t *tidx,
    IN uint64_t nelems,
    IN dmapp_type_t type,
    OUT dmapp_syncid_handle_t *syncid);
5.3.21.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer.

- **target_seg**
  Pointer to the segment descriptor of the target buffer.

- **target_pe**
  Target processing element.

- **source_addr**
  Pointer to the address of the source buffer.

- **tidx**
  Pointer to an array of positive offsets into target buffer. Offsets in the tidx array are in units of type. Each element to be transferred \( i \), where \( i = 1, \text{nelems} \) is transferred to \( \text{target_addr} + \text{tidx}(i) \). Note that the length of the array tidx should be equal to nelems, or a segmentation fault may occur.

- **nelems**
  Number of elements to be transferred.

- **type**
  Type of elements to be transferred.

- **syncid**
  Synchronization ID.

5.3.21.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_NO_SPACE**
  The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.22 dmapp_ixput_nbi

The dmapp_ixput_nbi function is a non-blocking implicit Indexed PUT.
Using the GNI and DMAPP APIs

5.3.22.1 Synopsis

```c
dmapp_return_t dmapp_ixput_nbi(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN void *source_addr,  
    IN ptrdiff_t *tidx,  
    IN uint64_t nelems,  
    IN dmapp_type_t type);
```

5.3.22.2 Parameters

- **target_addr**: Pointer to the address of the target buffer.
- **target_seg**: Pointer to the segment descriptor of the target buffer.
- **target_pe**: Target processing element.
- **source_addr**: Pointer to the address of the source buffer.
- **tidx**: Pointer to an array of positive offsets into the target buffer.
- **nelems**: Number of elements to transfer.
- **type**: Type of elements to transfer.

5.3.22.3 Return Codes

- **DMAPP_RC_SUCCESS**: The operation completed successfully.
- **DMAPP_RC_INVALID_PARAM**: One or more input parameters is invalid.
- **DMAPP_RC_NO_SPACE**: The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.

5.3.23 dmapp_ixput

The `dmapp_ixput` function is a blocking Indexed PUT.
5.3.23.1 Synopsis

```c
dmapp_return_t dmapp_iput(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN void *source_addr,
    IN ptrdiff_t *tidx,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

5.3.23.2 Parameters

- **target_addr**: Pointer to the address of the target buffer.
- **target_seg**: Pointer to a segment descriptor of the target buffer.
- **target_pe**: Target processing element.
- **source_addr**: Pointer to the address of the source buffer.
- **tidx**: Pointer to an array of positive offsets into the target buffer.
- **nelems**: Number of elements to transfer.
- **type**: Type of elements to transfer.

5.3.23.3 Return Codes

- **DMAPP_RC_SUCCESS**: The operation completed successfully.
- **DMAPP_RC_INVALID_PARAM**: One or more input parameters is invalid.
- **DMAPP_RC_NO_SPACE**: The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.
- **DMAPP_RC_TRANSACTION_ERROR**: A transaction error has occurred.

5.3.24 dmapp_ixget_nb

The `dmapp_ixget_nb` function is a non-blocking explicit Indexed GET.
GET data starting from a remote source address using offsets specified by the \texttt{sidx} array and returning the data into a contiguous block starting at address \texttt{target_addr} in local memory. The remote address is specified by the triplet virtual address \texttt{source_addr}, segment descriptor \texttt{source_seg} and source process \texttt{source_pe}. \texttt{nelems} specifies the number of elements of type \texttt{type} to be transferred. Offsets in the \texttt{sidx} array are in units of type. The memory region described by the remote address, \texttt{sidx} and \texttt{nelems} must reside in an exported memory of \texttt{source_pe}.

### 5.3.24.1 Synopsis

```c
dmapp_return_t dmapp_ixget_nb(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN ptrdiff_t *sidx,
    IN uint64_t nelems,
    IN dmapp_type_t type,
    OUT dmapp_syncid_handle_t *syncid);
```

### 5.3.24.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer.

- **source_addr**
  Pointer to the address of the source buffer.

- **source_seg**
  Pointer to the segment descriptor of the source buffer.

- **source_pe**
  Source processing element.

- **sidx**
  Pointer to an array of positive offsets into the source buffer.

- **nelems**
  Number of elements to transfer.

- **type**
  Type of elements to transfer. The DMAPP\_BYTE type is not supported.

- **syncid**
  Pointer to the synchronization ID.
5.3.24.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

The source or target buffer or length is not properly Dword (4 byte) aligned.

DMAPP_RC_RESOURCE_ERROR

A resource error occurred.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.25 dmapp_ixget_nbi

The dmapp_ixget_nbi function is a non-blocking implicit Indexed GET.

5.3.25.1 Synopsis

dmapp_return_t dmapp_ixget_nbi(  
  IN void *target_addr,  
  IN void *source_addr,  
  IN dmapp_seg_desc_t *source_seg,  
  IN dmapp_pe_t source_pe,  
  IN ptrdiff_t *sidx,  
  IN uint64_t nelems,  
  IN dmapp_type_t type);
5.3.25.2 Parameters

\textit{target\_addr}

Pointer to the address of the target buffer.

\textit{source\_addr}

Pointer to the address of the source buffer.

\textit{source\_seg}

Pointer to the segment descriptor of the source buffer.

\textit{source\_pe}

Source processing element.

\textit{sidx}

Pointer to an array of positive offsets into the source buffer.

\textit{nelems}

Number of elements to transfer.

\textit{type}

Type of elements to transfer. The type DMAPP\_BYTE is not supported.

5.3.25.3 Return Codes

DMAPP\_RC\_SUCCESS

The operation completed successfully.

DMAPP\_RC\_INVALID\_PARAM

One or more input parameters is invalid.

DMAPP\_RC\_ALIGNMENT\_ERROR

Source or target buffer or length not properly Dword (4 byte) aligned.

DMAPP\_RC\_RESOURCE\_ERROR

A resource error occurred.

DMAPP\_RC\_NO\_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max\textunderscore outstanding\textunderscore nb in the job attributes.

5.3.26 dmapp\_ixget

The dmapp\_ixget function is a blocking indexed GET.
5.3.26.1 Synopsis

```c
dmapp_return_t dmapp_ixget(
    IN void        *target_addr,
    IN void        *source_addr,
    IN dmapp_seg_desc_t  *source_seg,
    IN dmapp_pe_t          source_pe,
    IN ptrdiff_t        *sidx,
    IN uint64_t          nelems,
    IN dmapp_type_t       type);
```

5.3.26.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer.

- **source_addr**
  Pointer to the address of the source buffer.

- **source_seg**
  Pointer to the segment descriptor of the source buffer.

- **source_pe**
  Source processing element.

- **sidx**
  Pointer to an array of positive offsets into the source buffer.

- **nelems**
  Number of elements to transfer.

- **type**
  Type of elements to transfer. The DMAPP_BYTE type is not supported.

5.3.26.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_ALIGNMENT_ERROR**
  Source or target buffer or length not properly Dword (4 byte) aligned.

- **DMAPP_RC_RESOURCE_ERROR**
  A resource error occurred.
DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

DMAPP_RC_TRANSACTION_ERROR

A transaction error has occurred.

5.3.27 dmapp_put_ixpe_nb

The dmapp_put_ixpe_nb function is a non-blocking explicit PUT with indexed PE stride. It delivers data starting at source_addr in local memory to a list of target PEs target_pe_list starting at target_addr in their memories. nelems specifies the number of elements of type type to be PUT into each target PE. When the transfer is complete, each target PE will have a copy of the contents of the original source buffer. The address range specified by target_addr and nelems must reside in an exported, symmetric memory segment in each PE in target_pe_list.

5.3.27.1 Synopsis

dmapp_return_t dmapp_put_ixpe_nb(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t *target_pe_list,
    IN uint32_t num_target_pes,
    IN void *source_addr,
    IN uint64_t nelems,
    IN dmapp_type_t type,
    OUT dmapp_syncid_handle_t *syncid);
5.3.27.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer.

- **target_seg**
  Pointer to the segment descriptor of the target buffer.

- **target_pe_list**
  Pointer to the list of target processing elements.

- **num_target_pes**
  Number of target processing elements.

- **source_addr**
  Pointer to the address of the source buffer.

- **nelems**
  Number of elements to transfer.

- **type**
  Type of elements to transfer.

- **syncid**
  Pointer to the synchronization ID.

5.3.27.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_NO_SPACE**
  The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.

5.3.28 dmapp_put_ixpe_nbi

The `dmapp_put_ixpe_nbi` function is a non-blocking implicit PUT with indexed PE stride.

It delivers data starting at `source_addr` in local memory to a list of target PEs `target_pe_list` starting at `target_addr` in their memories. `nelems` specifies the number of elements of type `type` to be PUT into each target PE. When the transfer is complete, each target PE will have a copy of the contents of the original source buffer. The address range specified by `target_addr` and `nelems` must reside in an exported memory segment in each PE in `target_pe_list`.
5.3.28.1 Synopsis

```c
dmapp_return_t dmapp_put_ixpe_nbi(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t *target_pe_list,
    IN uint32_t num_target_pes,
    IN void *source_addr,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

5.3.28.2 Parameters

- **IN target_addr**
  Pointer to the address of the target buffer.

- **target_seg**
  Pointer to the segment descriptor of the target buffer.

- **target_pe_list**
  Pointer to a list of target processing elements.

- **num_target_pes**
  Number of target processing elements.

- **source_addr**
  Pointer to the address of the source buffer.

- **nelems**
  Number of elements to transfer.

- **type**
  Type of elements to transfer.

5.3.28.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_NO_SPACE**
  The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.
5.3.29 dmapp_put_ixpe

The dmapp_put_ixpe function is a blocking PUT with indexed PE stride. It delivers data starting at source_addr in local memory to a list of target PEs target_pe_list starting at target_addr in their memories. nelems specifies the number of elements of type type to be PUT into each target PE. When the transfer is complete, each target PE will have a copy of the contents of the original source buffer. The address range specified by target and nelems must reside in an exported memory segment in each PE in target_pe_list. The remote address is specified by the target virtual address target_addr and the segment descriptor target_seg. The address range specified by target_addr and nelems must reside in an exported, symmetric memory segment in each PE in target_pe_list.

5.3.29.1 Synopsis

```c
dmapp_return_t dmapp_put_ixpe(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t *target_pe_list,
    IN uint32_t num_target_pes,
    IN void *source_addr,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

5.3.29.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer.

- **target_seg**
  Pointer to the segment descriptor of the target buffer.

- **target_pe_list**
  Pointer to the list of target processing elements.

- **num_target_pes**
  Number of target processing elements.

- **source_addr**
  Pointer to the address of the source buffer.

- **nelems**
  Number of elements to transfer.

- **type**
  Type of elements to transfer.
5.3.29.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.

DMAPP_RC_TRANSACTION_ERROR

A transaction error has occurred.

5.3.30 dmapp_scatter_ixpe_nb

The `dmapp_scatter_ixpe_nb` function is a non-blocking explicit scatter with indexed PE stride. The function delivers data to a list of target PEs in the `target_pe_list` starting at `target_addr` in their memories. `nelems` specifies the number of elements of `type` to be PUT into each target PE. A remote PE at some index `I` in the `target_pe_list` will receive elements `I * nelems` to `(I+1) *nelems` - 1.

Unlike the `dmapp_put_ixpe` function, the `source_addr` array specifies a `num_target_pes * nelems * sizeof(type)` array.

The remote address is specified by the virtual address `target_addr` and segment descriptor `target_seg`. The address range specified by `target_addr` and `nelems` must reside in an exported, symmetric memory segment in each PE in `target_pe_list`.

5.3.30.1 Synopsis

```c
dmapp_return_t dmapp_scatter_ixpe_nb(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t *target_pe_list,
    IN uint32_t num_target_pes,
    IN void *source_addr,
    IN uint64_t nelems,
    IN dmapp_type_t type,
    OUT dmapp_syncid_handle_t *syncid);
```
5.3.30.2 Parameters

- **target_addr**: Pointer to an address of the target buffer.
- **target_seg**: Pointer to a segment descriptor of the target buffer.
- **target_pe_list**: Pointer to a list of target processing elements.
- **num_target_pes**: Number of target processing elements.
- **source_addr**: Pointer to the address of the source buffer.
- **nelems**: Number of elements to transfer.
- **type**: Type of elements to transfer.
- **syncid**: Pointer to the synchronization ID.

5.3.30.3 Return Codes

- **DMAPP_RC_SUCCESS**: The operation completed successfully.
- **DMAPP_RC_INVALID_PARAM**: One or more input parameters is invalid.
- **DMAPP_RC_NO_SPACE**: The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.

5.3.31 dmapp_scatter_ixpe_nbi

The `dmapp_scatter_ixpe_nbi` function is a non-blocking implicit scatter with indexed PE stride. The function delivers data to a list of target PEs in the `target_pe_list` starting at `target_addr` in their memories. `nelems` specifies the number of elements of type to be PUT into each target PE. A remote PE at some index I in the `target_pe_list` will receive elements \( I \times \text{nelems} \) to \( (I+1) \times \text{nelems} - 1 \).

Unlike the `dmapp_put_ixpe` function, the `source_addr` array specifies a `num_target_pes * nelems * sizeof(type)` array.
Using the GNI and DMAPP APIs

The remote address is specified by the virtual address `target_addr` and segment descriptor `target_seg`. The address range specified by `target_addr` and `nelems` must reside in an exported, symmetric memory segment in each PE in `target_pe_list`.

### 5.3.31.1 Synopsis

```c
dmapp_return_t dmapp_scatter_ixpe_nbi(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t *target_pe_list,
    IN uint32_t num_target_pes,
    IN void *source_addr,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

### 5.3.31.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer.

- **target_seg**
  Pointer to the segment descriptor of the target buffer.

- **target_pe_list**
  Pointer to the list of target processing elements.

- **num_target_pes**
  Number of target processing elements.

- **source_addr**
  Pointer to the address of the source buffer.

- **nelems**
  Number of elements to transfer.

- **type**
  Type of elements to transfer.

### 5.3.31.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_NO_SPACE**
  The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.
5.3.32 dmapp_scatter_ixpe

The dmapp_scatter_ixpe function is a blocking scatter with indexed PE stride. The function delivers data to a list of target PEs in the target_pe_list starting at target_addr in their memories. nelems specifies the number of elements of type to be PUT into each target PE.

A remote PE at some index I in the target_pe_list will receive elements I * nelems to (I+1) *nelems-1. The address range specified by target_addr and nelems must reside in an exported memory segment in each PE specified in target_pe_list.

Unlike the dmapp_put_ixpe function, the source_addr array specifies a num_target_pes * nelems * sizeof(type) array.

The remote address is specified by the virtual address target_addr and segment descriptor target_seg. The address range specified by target_addr and nelems must reside in an exported, symmetric memory segment in each PE in target_pe_list.

5.3.32.1 Synopsis

```c
dmapp_return_t dmapp_scatter_ixpe(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t *target_pe_list,
    IN uint32_t num_target_pes,
    IN void *source_addr,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

5.3.32.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer.

- **target_seg**
  Pointer to the segment descriptor of the target buffer.

- **target_pe_list**
  Pointer to the list of target processing elements.

- **num_target_pes**
  Number of target processing elements.

- **source_addr**
  Pointer to the address of the source buffer.

- **nelems**
  Number of elements to transfer.

- **type**
  Type of elements to transfer.
5.3.32.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.33 dmapp_gather_ixpe_nb

The dmapp_gather_ixpe_nb function is a non-blocking explicit gather with indexed PE stride. Gather data starting at source_addr from the list of PEs specified by source_pe_list and concatenate the returned data in a buffer in local memory specified by target_addr. nelems specifies the number of elements of type collected from each PE.

The address range specified by source_addr and nelems must reside in an exported, symmetric memory segment in each PE in source_pe_list.

5.3.33.1 Synopsis

dmapp_return_t dmapp_gather_ixpe_nb(  
    IN void *target_addr,  
    IN void *source_addr,  
    IN dmapp_seg_desc_t *source_seg,  
    IN dmapp_pe_t *source_pe_list,  
    IN uint32_t num_source_pes,  
    IN uint64_t nelems,  
    IN dmapp_type_t type,  
    OUT dmapp_syncid_handle_t *syncid);
5.3.33.2 Parameters

\textit{target\_addr}  
Pointer to the address of the target buffer.

\textit{source\_addr}  
Pointer to the address of the source buffer.

\textit{source\_seg}  
Pointer to the segment descriptor of the source buffer.

\textit{source\_pe\_list}  
Pointer to the list of source processing elements.

\textit{num\_source\_pes}  
Number of source processing elements.

\textit{nelems}  
Number of elements to transfer.

\textit{type}  
Type of elements to transfer.

\textit{syncid}  
Returns a pointer to the synchronization ID.

5.3.33.3 Return Codes

DMAPP\_RC\_SUCCESS  
The operation completed successfully.

DMAPP\_RC\_INVALID\_PARAM  
One or more input parameters is invalid.

DMAPP\_RC\_NO\_SPACE  
The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for \texttt{max\_outstanding\_nb} in the job attributes.

5.3.34 \texttt{dmapp\_gather\_ixpe\_nbi}

The \texttt{dmapp\_gather\_ixpe\_nbi} function is a non-blocking implicit gather with indexed PE stride. Gather data starting at \textit{source\_addr} from the list of PEs specified by \textit{source\_pe\_list} and concatenate the returned data in a buffer in local memory specified by \textit{target\_addr}. \texttt{nelems} specifies the number of elements of \textit{type} collected from each PE.

The address range specified by \textit{source\_addr} and \texttt{nelems} must reside in an exported, symmetric memory segment in each PE in \textit{source\_pe\_list}. 
5.3.34.1 Synopsis

```c
dmapp_return_t dmapp_gather_ixpe_nbi(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t *source_pe_list,
    IN uint32_t num_source_pes,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

5.3.34.2 Parameters

- `target_addr`  
  Pointer to the address of the target buffer.
- `source_addr`  
  Pointer to the address of the source buffer.
- `source_seg`  
  Pointer to the segment descriptor of the source buffer.
- `source_pe_list`  
  Pointer to the list of source processing elements.
- `num_source_pes`  
  Number of source processing elements.
- `nelems`  
  Number of elements to transfer.
- `type`  
  Type of elements to transfer.

5.3.34.3 Return Codes

- `DMAPP_RC_SUCCESS`  
  The operation completed successfully.
- `DMAPP_RC_INVALID_PARAM`  
  One or more input parameters is invalid.
- `DMAPP_RC_NO_SPACE`  
  The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.
5.3.35 dmapp_gather_ixpe

The dmapp_gather_ixpe function is a blocking gather with indexed processing element stride. Gather data starting at source_addr from the list of PEs specified by source_pe_list and concatenate the returned data in a buffer in local memory specified by target_addr. nelems specifies the number of elements of type collected from each PE.

The address range specified by source_addr and nelems must reside in an exported, symmetric memory segment in each PE listed in source_pe_list.

5.3.35.1 Synopsis

```c
dmapp_return_t dmapp_gather_ixpe(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t *source_pe_list,
    IN uint32_t num_source_pes,
    IN uint64_t nelems,
    IN dmapp_type_t type);
```

5.3.35.2 Parameters

- **target_addr**
  
  Pointer to the address of the target buffer.

- **source_addr**
  
  Pointer to the address of the source buffer.

- **source_seg**
  
  Pointer to the segment descriptor of the source buffer.

- **source_pe_list**
  
  Pointer to the list of source processing elements.

- **num_source_pes**
  
  Number of source processing elements.

- **nelems**
  
  Number of elements to transfer.

- **type**
  
  Type of elements to transfer.
5.3.35.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for \texttt{max\_outstanding\_nb} in the job attributes.

DMAPP_RC_TRANSACTION_ERROR

A transaction error has occurred.

5.3.36 \texttt{dmapp\_aadd\_qw\_nb}

The \texttt{dmapp\_aadd\_qw\_nb} function is a non-blocking explicit atomic ADD.

5.3.36.1 Synopsis

\begin{verbatim}
dmapp_return_t dmapp_aadd_qw_nb(
    IN void * target_addr,
    IN dmapp_seg_desc_t * target_seg,
    IN dmapp_pe_t target_pe,
    IN int64_t operand,
    OUT dmapp_syncid_handle_t * syncid);
\end{verbatim}

5.3.36.2 Parameters

\begin{itemize}
  \item \textit{target\_addr}  
    Pointer to the address of the target buffer (for Qword only).
  \item \textit{target\_seg}  
    Pointer to the segment descriptor for the target buffer.
  \item \textit{target\_pe}  
    Target processing element.
  \item \textit{operand}  
    Value to be added.
  \item \textit{syncid}  
    Pointer to the synchronization ID.
\end{itemize}
5.3.36.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.37 dmapp_aadd_qw_nbi

The dmapp_aadd_qw_nbi function is a non-blocking implicit atomic ADD.

5.3.37.1 Synopsis

dmapp_return_t dmapp_aadd_qw_nbi(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN int64_t operand);

5.3.37.2 Parameters

target_addr

Pointer to the address of the target buffer (Qword only).

target_seg

Pointer to the segment descriptor for the target buffer.

target_pe

Target processing element.

operand

Value to be added.
5.3.37.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.38 dmapp_aadd_qw

The dmapp_aadd_qw function is a blocking atomic ADD.

5.3.38.1 Synopsis

```
dmapp_return_t dmapp_aadd_qw(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN int64_t operand);
```

5.3.38.2 Parameters

- **target_addr**
  
  Pointer to the address of the target buffer (Qword only).

- **target_seg**
  
  Pointer to the segment descriptor for the target buffer.

- **target_pe**
  
  Target processing element.

- **operand**
  
  Value to be added.
5.3.38.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

DMAPP_RC_TRANSACTION_ERROR

A transaction error has occurred.

5.3.39 dmapp_aand_qw_nb

The dmapp_aand_qw_nb function is a non-blocking explicit atomic AND.

5.3.39.1 Synopsis

dmapp_return_t dmapp_aand_qw_nb(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN int64_t operand,
    OUT dmapp_syncid_handle_t *syncid);

5.3.39.2 Parameters

*target_addr

Pointer to the address of the target buffer (Qword only).

*target_seg

Pointer to the segment descriptor for the target buffer.

target_pe

Target processing element.

operand

Operand for the AND operation.

*syncid

Pointer to the synchronization ID.
5.3.39.3 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_INVALID_PARAM
One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR
Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE
The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.40 dmapp_aand_qw_nbi

The dmapp_aand_qw_nbi function is a non-blocking implicit atomic AND.

5.3.40.1 Synopsis

dmapp_return_t dmapp_aand_qw_nbi(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN int64_t operand);

5.3.40.2 Parameters

 target_addr
    Pointer to the address of the target buffer (Qword only).

 target_seg
    Pointer to the segment descriptor for the target buffer.

 target_pe
    Target processing element.

 operand
    Operand for the AND operation.
5.3.40.3 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_INVALID_PARAM
One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR
Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE
The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.41 dmapp_aand_qw
The dmapp_aand_qw function is a blocking atomic AND.

5.3.41.1 Synopsis

```c
dmapp_return_t dmapp_aand_qw(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN int64_t operand);
```

5.3.41.2 Parameters

- **target_addr**
  Pointer the address of the target buffer (Qword only).

- **target_seg**
  Pointer to the segment descriptor for the target buffer.

- **target_pe**
  Target processing element.

- **operand**
  Operand for the AND operation.
5.3.41.3 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_INVALID_PARAM
One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR
Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE
The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

DMAPP_RC_TRANSACTION_ERROR
A transaction error has occurred.

5.3.42 dmapp_aor_qw_nb

The dmapp_aor_qw_nb function is a non-blocking explicit atomic OR.

5.3.42.1 Synopsis

```c
dmapp_return_t dmapp_aor_qw_nb(
    IN  void       *target_addr,
    IN  dmapp_seg_desc_t  *target_seg,
    IN  dmapp_pe_t       target_pe,
    IN  int64_t         operand,
    OUT dmapp_syncid_handle_t *syncid);
```

5.3.42.2 Parameter

`target_addr`
Pointer to the address of the target buffer (Qword only).

`target_seg`
Pointer to the segment descriptor for the target buffer.

`target_pe`
Target processing element.

`operand`
Operand for the OR operation.

`syncid`
Pointer to the synchronization ID.
5.3.42.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.43 dmapp_aor_qw_nbi

The dmapp_aor_qw_nbi function is a non-blocking implicit atomic OR.

5.3.43.1 Synopsis

dmapp_return_t dmapp_aor_qw_nbi(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN int64_t operand);

5.3.43.2 Parameter

    target_addr

        Pointer to the address of the target buffer (Qword only).

    target_seg

        Pointer to the segment descriptor for the target buffer.

    target_pe

        Target processing element.

    operand

        Operand for the OR operation.
5.3.43.3 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_INVALID_PARAM
One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR
Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE
The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.44 dmapp_aor_qw

The dmapp_aor_qw function is a blocking atomic OR.

5.3.44.1 Synopsis

```c
dmapp_return_t dmapp_aor_qw(
   IN void * target_addr,
   IN dmapp_seg_desc_t * target_seg,
   IN dmapp_pe_t target_pe,
   IN int64_t operand);
```

5.3.44.2 Parameters

- **target_addr**
  - Pointer to the address of the target buffer (Qword only).

- **target_seg**
  - Pointer to the segment descriptor for the target buffer.

- **target_pe**
  - Target processing element.

- **operand**
  - Operand for the OR operation.
5.3.44.3 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_INVALID_PARAM
One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR
Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE
The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.45 dmapp_axor_qw_nb

The dmapp_axor_qw_nb function is a non-blocking explicit atomic XOR.

5.3.45.1 Synopsis

dmapp_return_t dmapp_axor_qw_nb(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN int64_t operand,
    OUT dmapp_syncid_handle_t *syncid);

5.3.45.2 Parameters

target_addr
Pointer to the address of the target buffer (Qword only).

target_seg
Pointer to the segment descriptor for the target buffer.

target_pe
Target processing element.

operand
Operand for the XOR operation.

syncid
Pointer to the synchronization ID.
5.3.45.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.46 dmapp_axor_qw_nbi

The dmapp_axor_qw_nbi function is a non-blocking implicit atomic XOR.

5.3.46.1 Synopsis

dmapp_return_t dmapp_axor_qw_nbi(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN int64_t operand);

5.3.46.2 Parameter

    target_addr
    Pointer to the address of the target buffer (Qword only).

    target_seg
    Pointer to the segment descriptor for the target buffer.

    target_pe
    Target processing element.

    operand
    Operand for the XOR operation.
5.3.46.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.47 dmapp_axor_qw

The dmapp_axor_qw function is a blocking atomic XOR.

5.3.47.1 Synopsis

dmapp_return_t dmapp_axor_qw(
    IN void *target_addr,
    IN dmapp_seg_desc_t *target_seg,
    IN dmapp_pe_t target_pe,
    IN int64_t operand);

5.3.47.2 Parameters

target_addr

Pointer to the address of the target buffer (Qword only).

target_seg

Pointer to the segment descriptor for the target buffer.

target_pe

Target processing element.

operand

Operand for an XOR operation.
5.3.47.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

DMAPP_RC_TRANSACTION_ERROR

A transaction error has occurred.

5.3.48 dmapp_afadd_qw_nb

The dmapp_afadd_qw_nb function is a non-blocking explicit atomic FADD.

5.3.48.1 Synopsis

```
dmapp_return_t dmapp_afadd_qw_nb(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN int64_t operand,
    OUT dmapp_syncid_handle_t *syncid);
```
5.3.48.2 Parameters

*target_addr*

Pointer to the address of the target buffer where the result is returned (Qword only).

*source_addr*

Pointer to the address of the source buffer (Qword only).

*source_seg*

Pointer to the segment descriptor of the source buffer.

*source_pe*

Source processing element.

*operand*

Operand for the FADD operation.

*syncid*

Pointer to the synchronization ID.

5.3.48.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

5.3.49 dmapp_afadd_qw_nbi

The dmapp_afadd_qw_nbi function is a non-blocking implicit atomic FADD.

5.3.49.1 Synopsis

```c
dmapp_return_t dmapp_afadd_qw_nbi(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN int64_t operand);
```
5.3.49.2 Parameters

*target_addr*

Pointer to the address of a target buffer where the result is returned (Qword only).

*source_addr*

Pointer to the address of the source buffer (Qword only).

*source_seg*

Pointer to the segment descriptor of the source buffer.

*source_pe*

Source processing element.

*operand*

Operand for the FADD operation.

5.3.49.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.50 dmapp_afadd_qw

The dmapp_afadd_qw function is a blocking atomic FADD.

5.3.50.1 Synopsis

```c
dmapp_return_t dmapp_afadd_qw(  
    IN void *target_addr,  
    IN void *source_addr,  
    IN dmapp_seg_desc_t *source_seg,  
    IN dmapp_pe_t source_pe,  
    IN int64_t operand);  
```
5.3.50.2 Parameters

\textit{target\_addr}

Pointer to the address of the target buffer where the result is returned (Qword only).

\textit{source\_addr}

Pointer to the address of the source buffer (Qword only).

\textit{source\_seg}

Pointer to the segment descriptor of the source buffer.

\textit{source\_pe}

Source processing element.

\textit{operand}

Operand for the FADD operation.

5.3.50.3 Return Codes

\textbf{DMAPP\_RC\_SUCCESS}

The operation completed successfully.

\textbf{DMAPP\_RC\_INVALID\_PARAM}

One or more input parameters is invalid.

\textbf{DMAPP\_RC\_ALIGNMENT\_ERROR}

Target buffer not properly (Qword, 8 byte) aligned.

\textbf{DMAPP\_RC\_NO\_SPACE}

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max\_outstanding\_nb in the job attributes.

\textbf{DMAPP\_RC\_ TRANSACTION\_ERROR}

A transaction error has occurred.

5.3.51 \texttt{dmapp\_afand\_qw\_nb}

The \texttt{dmapp\_afand\_qw\_nb} function is a non-blocking explicit atomic FAND.

5.3.51.1 Synopsis

\begin{verbatim}
\texttt{dmapp\_return\_t dmapp\_afand\_qw\_nb(}
\texttt{    IN void \quad \ast \textit{target\_addr},
\texttt{    IN void \quad \ast \textit{source\_addr},
\texttt{    IN dmapp\_seg\_desc\_t \quad \ast \textit{source\_seg},
\texttt{    IN dmapp\_pe\_t \quad \textit{source\_pe},
\texttt{    IN int64\_t \quad \textit{operand},
\texttt{    OUT dmapp\_syncid\_handle\_t \quad \ast \textit{syncid});}
\end{verbatim}
5.3.51.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer where the result is returned (Qword only).

- **source_addr**
  Pointer to the address of the source buffer (Qword only).

- **source_seg**
  Pointer to the segment descriptor of the source buffer.

- **source_pe**
  Source processing element.

- **operand**
  Operand for the FAND operation.

- **syncid**
  Pointer to the synchronization ID.

5.3.51.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_ALIGNMENT_ERROR**
  Target buffer not properly (Qword, 8 byte) aligned.

- **DMAPP_RC_NO_SPACE**
  The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.

5.3.52 `dmapp_afand_qw_nbi`

The `dmapp_afand_qw_nbi` function is a non-blocking implicit atomic FAND.

5.3.52.1 Synopsis

```c
dmapp_return_t
dmapp_afand_qw_nbi(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN int64_t operand);
```
5.3.52.2 Parameters

*target_addr*

Pointer to the address of the target buffer where the result is returned (Qword only).

*source_addr*

Pointer to the address of the source buffer (Qword only).

*source_seg*

Segment descriptor of source buffer.

*source_pe*

Source processing element.

*operand*

Operand for an FAND operation.

5.3.52.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.

5.3.53 dmapp_afand_qw

The `dmapp_afand_qw` function is a blocking atomic FAND.

5.3.53.1 Synopsis

```c
dmapp_return_t dmapp_afand_qw(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN int64_t operand);
```
5.3.53.2 Parameters

- **target_add**: Pointer to the address of the target buffer where the result is returned (Qword only).
- **source_addr**: Pointer to the address of the source buffer (Qword only).
- **source_seg**: Pointer to the segment descriptor of the source buffer.
- **source_pe**: Source processing element.
- **operand**:Operand for an \texttt{FAND} operation.

5.3.53.3 Return Codes

- **DMAPP\_RC\_SUCCESS**: The operation completed successfully.
- **DMAPP\_RC\_INVALID\_PARAM**: One or more input parameters is invalid.
- **DMAPP\_RC\_ALIGNMENT\_ERROR**: Target buffer not properly (Qword, 8 byte) aligned.
- **DMAPP\_RC\_NO\_SPACE**: The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for \texttt{max\_outstanding\_nb} in the job attributes.

5.3.54 \texttt{dmapp\_afxor\_qw\_nb}

The \texttt{dmapp\_afxor\_qw\_nb} function is a non-blocking explicit atomic FXOR.

5.3.54.1 Synopsis

```c
dmapp_return_t dmapp_afxor_qw_nb(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN int64_t operand,
    OUT dmapp_syncid_handle_t *syncid);
```
5.3.54.2 Parameters

**target_addr**
Pointer to the address of the target buffer where the result is returned (Qword only).

**source_addr**
Pointer to the address of the source buffer (Qword only).

**source_seg**
Pointer to the segment descriptor of the source buffer.

**source_pe**
Source processing element.

**operand**
Operand for an FXOR operation.

**syncid**
Pointer to the synchronization ID.

5.3.54.3 Return Codes

**DMAPP_RC_SUCCESS**
The operation completed successfully.

**DMAPP_RC_INVALID_PARAM**
One or more input parameters is invalid.

**DMAPP_RC_ALIGNMENT_ERROR**
Target buffer not properly (Qword, 8 byte) aligned.

**DMAPP_RC_NO_SPACE**
The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for *max_outstanding_nb* in the job attributes.

5.3.55 dmapp_afxor_qw_nbi

The *dmapp_afxor_qw_nbi* function is a non-blocking implicit atomic FXOR.

5.3.55.1 Synopsis

```c
dmapp_return_t dmapp_afxor_qw_nbi(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN int64_t operand);
```
5.3.55.2 Parameters

\textit{target_addr}

Pointer to the address of the target buffer where the result is returned (Qword only).

\textit{source_addr}

Pointer to the address of the source buffer (Qword only).

\textit{source_seg}

Pointer to the segment descriptor of the source buffer.

\textit{source_pe}

Source processing element.

\textit{operand}

Operand for an \texttt{FXOR} operation.

5.3.55.3 Return Codes

\texttt{DMAPP_RC_SUCCESS}

The operation completed successfully.

\texttt{DMAPP_RC_INVALID_PARAM}

One or more input parameters is invalid.

\texttt{DMAPP_RC_ALIGNMENT_ERROR}

Target buffer not properly (Qword, 8 byte) aligned.

\texttt{DMAPP_RC_NO_SPACE}

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for \texttt{max\_outstanding\_nb} in the job attributes.

5.3.56 \texttt{dmapp\_afxor\_qw}

The \texttt{dmapp\_afxor\_qw} function is a blocking atomic \texttt{FXOR}.

5.3.56.1 Synopsis

\begin{verbatim}
dmapp_return_t dmapp_afxor_qw(
    IN void * target_addr,
    IN void * source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN int64_t operand);
\end{verbatim}
5.3.56.2 Parameters

\textit{target\_addr}

Pointer to the address of the target buffer where the result is returned (Qword only).

\textit{source\_addr}

Pointer to the address of the source buffer (Qword only).

\textit{source\_seg}

Pointer to the segment descriptor of the source buffer.

\textit{source\_pe}

Source processing element.

\textit{operand}

Operand for an FXOR operation.

5.3.56.3 Return Codes

\textbf{DMAPP\_RC\_SUCCESS}

The operation completed successfully.

\textbf{DMAPP\_RC\_INVALID\_PARAM}

One or more input parameters is invalid.

\textbf{DMAPP\_RC\_ALIGNMENT\_ERROR}

Target buffer not properly (Qword, 8 byte) aligned.

\textbf{DMAPP\_RC\_NO\_SPACE}

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max\_outstanding\_nb in the job attributes.

\textbf{DMAPP\_RC\_TRANSACTION\_ERROR}

A transaction error has occurred.

5.3.57 \texttt{dmapp\_afor\_qw\_nb}

The \texttt{dmapp\_afor\_qw\_nb} function is a non-blocking explicit atomic FOR.

5.3.57.1 Synopsis

\begin{verbatim}
dmapp_return_t dmapp_afor_qw_nb(
  IN void *target_addr,
  IN void *source_addr,
  IN dmapp_seg_desc_t *source_seg,
  IN dmapp_pe_t source_pe,
  IN int64_t operand,
  OUT dmapp_syncid_handle_t *syncid);
\end{verbatim}
5.3.57.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer where the result is returned (Qword only).

- **source_addr**
  Pointer to the address of the source buffer (Qword only).

- **source_seg**
  Pointer to the segment descriptor of the source buffer.

- **source_pe**
  Source processing element.

- **operand**
  Operand for a FOR operation.

- **syncid**
  Pointer to the synchronization ID.

5.3.57.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_ALIGNMENT_ERROR**
  Target buffer not properly (Qword, 8 byte) aligned.

- **DMAPP_RC_NO_SPACE**
  The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.58 dmapp_afor_qw_nbi

The dmapp_afor_qw_nbi function is a non-blocking implicit atomic FOR.

5.3.58.1 Synopsis

```c
dmapp_return_t dmapp_afor_qw_nbi(
    IN void  *target_addr,
    IN void   *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t        source_pe,
    IN int64_t           operand);
```
5.3.58.2 Parameters

*target_addr*

Pointer to the address of the target buffer where the result is returned (Qword only).

*source_addr*

Pointer to the address of the source buffer (Qword only).

*source_seg*

Pointer to the segment descriptor of the source buffer.

*source_pe*

Source processing element.

*operand*

Operand for a FOR operation.

5.3.58.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.59 dmapp_afor_qw

The dmapp_afor_qw function is a blocking atomic FOR.

5.3.59.1 Synopsis

```c
dmapp_return_t dmapp_afor_qw(  
    IN void *target_addr,  
    IN void *source_addr,  
    IN dmapp_seg_desc_t *source_seg,  
    IN dmapp_pe_t source_pe,  
    IN int64_t operand);  
```
5.3.59.2 Parameters

\[ target\_addr \]

Pointer to the address of the target buffer where the result is returned (Qword only).

\[ source\_addr \]

Pointer to the address of the source buffer (Qword only).

\[ source\_seg \]

Pointer to the segment descriptor of the source buffer.

\[ source\_pe \]

Source processing element.

\[ operand \]

Operand for a FOR operation.

5.3.59.3 Return Codes

DMAPP\_RC\_SUCCESS

The operation completed successfully.

DMAPP\_RC\_INVALID\_PARAM

One or more input parameters is invalid.

DMAPP\_RC\_ALIGNMENT\_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

DMAPP\_RC\_NO\_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max\_outstanding\_nb in the job attributes.

DMAPP\_RC\_TRANSACTION\_ERROR

A transaction error has occurred.

5.3.60 dmapp\_afax\_qw\_nb

The dmapp\_afax\_qw\_nb function is a non-blocking explicit atomic FAX.
5.3.60.1 Synopsis

    dmapp_return_t dmapp_afax_qw_nb(
        IN void *
            *target_addr,
        IN void *
            *source_addr,
        IN dmapp_seg_desc_t *
            *source_seg,
        IN dmapp_pe_t
            source_pe,
        IN int64_t
            andMask,
        IN int64_t
            xorMask,
        OUT dmapp_syncid_handle_t *
            syncid);

5.3.60.2 Parameters

    target_addr
        Pointer to the address of the target buffer where the result is returned
        (Qword only).

    source_addr
        Pointer to the address of the source buffer (Qword only).

    source_seg
        Pointer to the segment descriptor of the source buffer.

    source_pe
        Source processing element.

    andMask
        Mask for an AND operation.

    xorMask
        Mask for an XOR operation.

    syncid
        Pointer to the synchronization ID.

5.3.60.3 Return Codes

    DMAPP_RC_SUCCESS
        The operation completed successfully.

    DMAPP_RC_INVALID_PARAM
        One or more input parameters is invalid.

    DMAPP_RC_ALIGNMENT_ERROR
        Target buffer not properly (Qword, 8 byte) aligned.

    DMAPP_RC_NO_SPACE
        The transaction request could not be completed due to insufficient
        resources. To resolve this error, synchronize more often, or if
        possible, increase the value for max_outstanding_nb in the
        job attributes.
5.3.61 dmapp_afax_qw_nbi

The dmapp_afax_qw_nbi function is a non-blocking implicit atomic FAX.

5.3.61.1 Synopsis

```c
dmapp_return_t dmapp_afax_qw_nbi(
    IN void * target_addr,
    IN void * source_addr,
    IN dmapp_seg_desc_t * source_seg,
    IN dmapp_pe_t source_pe,
    IN int64_t andMask,
    IN int64_t xorMask);
```

5.3.61.2 Parameters

- **target_addr**
  Pointer to the address of the target buffer where the result is returned (Qword only).

- **source_addr**
  Pointer to the address of the source buffer (Qword only).

- **source_seg**
  Pointer to the segment descriptor of the source buffer.

- **source_pe**
  Source processing element.

- **andMask**
  Mask for an AND operation.

- **xorMask**
  Mask for an XOR operation.

5.3.61.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_ALIGNMENT_ERROR**
  Target buffer not properly (Qword, 8 byte) aligned.

- **DMAPP_RC_NO_SPACE**
  The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for `max_outstanding_nb` in the job attributes.
5.3.62 dmapp_afax_qw

The dmapp_afax_qw function is a blocking atomic FAX.

5.3.62.1 Synopsis

```c
dmapp_return_t dmapp_afax_qw(
    IN void * target_addr,
    IN void * source_addr,
    IN dmapp_seg_desc_t * source_seg,
    IN dmapp_pe_t source_pe,
    IN int64_t andMask,
    IN int64_t xorMask);
```

5.3.62.2 Parameters

- `target_addr`  
  Pointer to the address of the target buffer where the result is returned (Qword only).

- `source_addr`  
  Pointer to the address of the source buffer (Qword only).

- `source_seg`  
  Pointer to the segment descriptor of the source buffer.

- `source_pe`  
  Source processing element.

- `andMask`  
  Mask for an AND operation.

- `xorMask`  
  Mask for an XOR operation.

5.3.62.3 Return Codes

- `DMAPP_RC_SUCCESS`  
  The operation completed successfully.

- `DMAPP_RC_INVALID_PARAM`  
  One or more input parameters is invalid.

- `DMAPP_RC_ALIGNMENT_ERROR`  
  Target buffer not properly (Qword, 8 byte) aligned.
DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

DMAPP_RC_TRANSACTION_ERROR

A transaction error has occurred.

5.3.63 dmapp_acswap_qw_nb

The dmapp_acswap_qw_nb function is a non-blocking explicit atomic CSWAP.

5.3.63.1 Synopsis

```c
dmapp_return_t dmapp_acswap_qw_nb(
    IN void * target_addr,
    IN void * source_addr,
    IN dmapp_seg_desc_t * source_seg,
    IN dmapp_pe_t source_pe,
    IN int64_t comperand,
    IN int64_t swaperand,
    OUT dmapp_syncid_handle_t * syncid);
```

5.3.63.2 Parameters

- **target_addr**
  
  Pointer to the address of the target buffer where the result is returned (Qword only).

- **source_addr**
  
  Pointer to the address of the source buffer (Qword only).

- **source_seg**
  
  Pointer to the segment descriptor of the source buffer.

- **source_pe**
  
  Source processing element.

- **comperand**
  
  Operand against which to compare.

- **swaperand**
  
  Operand to swap in.

- **syncid**
  
  Pointer to a synchronization ID.
5.3.63.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.64 dmapp_acswap_qw_nbi

The dmapp_acswap_qw_nbi function is a non-blocking implicit atomic CSWAP.

5.3.64.1 Synopsis

```
dmapp_return_t dmapp_acswap_qw_nbi(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN int64_t comperand,
    IN int64_t swaperand);
```

5.3.64.2 Parameters

target_addr

Pointer to the address of the target buffer where the result is returned (Qword only).

source_addr

Pointer to the address of the source buffer (Qword only).

source_seg

Pointer to the segment descriptor of the source buffer.

source_pe

Source processing element.

comperand

Operand against which to compare.

swaperand

Operand to swap in.
5.3.64.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR

Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE

The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

5.3.65 dmapp_acswap_qw

The dmapp_acswap_qw function is a blocking atomic CSWAP.

5.3.65.1 Synopsis


dmapp_return_t dmapp_acswap_qw(
    IN void *target_addr,
    IN void *source_addr,
    IN dmapp_seg_desc_t *source_seg,
    IN dmapp_pe_t source_pe,
    IN int64_t comperand,
    IN int64_t swaperand);

5.3.65.2 Parameters

    target_addr

    Pointer to the address of the target buffer where the result is returned (Qword only).

    source_addr

    Pointer to the address of the source buffer (Qword only).

    source_seg

    Pointer to the segment descriptor of the source buffer.

    source_pe

    Source processing element.

    comperand

   Operand against which to compare.

    swaperand

Operand to swap in.
5.3.65.3 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_INVALID_PARAM
One or more input parameters is invalid.

DMAPP_RC_ALIGNMENT_ERROR
Target buffer not properly (Qword, 8 byte) aligned.

DMAPP_RC_NO_SPACE
The transaction request could not be completed due to insufficient resources. To resolve this error, synchronize more often, or if possible, increase the value for max_outstanding_nb in the job attributes.

DMAPP_RC_TRANSACTION_ERROR
A transaction error has occurred.

5.3.66 dmapp_syncid_test
The dmapp_syncid_test function tests syncid for completion. It sets flag to 1 if the remote memory accesses associated with syncid are globally visible in the system. If the RMA request associated with the syncid has not completed, flag is set to 0.

5.3.66.1 Synopsis

dmapp_return_t dmapp_syncid_test(
    INOUT dmapp_syncid_handle_t *syncid,
    OUT int *flag);

5.3.66.2 Parameters

syncid Pointer to the syncid to test for completion.

flag Pointer to the flag indicating global visibility.
5.3.66.3 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_INVALID_PARAM
One or more input parameters is invalid.

DMAPP_RC_TRANSACTION_ERROR
A transaction error has occurred.

5.3.67 dmapp_syncid_wait
The dmapp_syncid_wait function is a wait for completion of request associated with syncid.

5.3.67.1 Synopsis

dmapp_return_t dmapp_syncid_wait(
    INOUT dmapp_syncid_handle_t *syncid);

5.3.67.2 Parameters

syncid
The syncid to test for completion.

5.3.67.3 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_INVALID_PARAM
One or more input parameters is invalid.

DMAPP_RC_TRANSACTION_ERROR
A transaction error has occurred.

5.3.68 dmapp_gsync_test
The dmapp_gsync_test function is a test for completion of issued nb implicit requests. It sets flag to 1 if remote memory accesses associated with previously issued non-blocking implicit RMA requests are globally visible in the system. Otherwise, flag is set to 0.
5.3.68 Synopsis

```c
dmapp_return_t dmapp_gsync_test(
    OUT int *flag);
```

5.3.68.2 Parameters

flag Pointer to a flag indicating global visibility.

5.3.68.3 Return Codes

- DMAPP_RC_SUCCESS
  The operation completed successfully.
- DMAPP_RC_TRANSACTION_ERROR
  A transaction error has occurred.

5.3.69 dmapp_gsync_wait

The `dmapp_gsync_wait` function forces a wait for completion of issued nb implicit requests. This is the blocking version of `dmapp_gsync_test`. The function only returns when all remote memory accesses associated with previously issued non-blocking implicit RMA requests are globally visible in the system.

5.3.69.1 Synopsis

```c
dmapp_return_t dmapp_gsync_wait(void);
```

5.3.69.2 Return Codes

- DMAPP_RC_SUCCESS
  The operation completed successfully.
- DMAPP_RC_TRANSACTION_ERROR
  A transaction error has occurred.

5.3.70 dmapp_c_pset_create

5.3.70.1 Synopsis

```c
dmapp_return_t dmapp_c_pset_create(
    IN dmapp_c_pset_desc_t *pdesc,
    IN uint64_t identifier,
    IN uint64_t modes,
    IN dmapp_c_pset_attrs_t *attrs,
    OUT dmapp_c_pset_handle_t *pset_handle);
```
5.3.70.2 Parameters

- **pdesc**: Pointer to a `dmapp_c_pset_desc_t` structure previously initialized to describe the ranks contained in the `pset`.

- **identifier**: Unique, nonzero 64-bit identifier to be associated with this `pset`. The same value must be supplied by all ranks creating the `pset`.

- **modes**: Specifies additional information about the `pset`. Concatenate operations on `a` are allowed when the `DMAPP_C_PSET_MODE_CONCAT` mode bit is set.

- **attrs**: Pointer to a previously initialized attribute structure. Can be `NULL`.

- **pset_handle**: Pointer to an opaque `dmapp_c_pset_handle_t` structure to be used for subsequent DMAPP collective calls.

5.3.70.3 Return Codes

- **DMAPP_RC_SUCCESS**: The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**: One or more input parameters is invalid.

- **DMAPP_RC_NO_SPACE**: Too many `psets` have already been created.

- **DMAPP_RC_BUSY**: The specified identifier is already in use.

5.3.71 `dmapp_c_pset_export`

Export a `pset` before calling DMAPP collective functions. The application must use some out-of-band synchronization mechanism after calling `dmapp_c_pset_create` and before invoking this function. Without this synchronization, there is a high probability that for one or more ranks, this function will return `DMAPP_RC_NOT_FOUND`.

This function will be called by all ranks in a job which are going to be used in the collective operations involving the `pset`.
All ranks must provide the same unique identifier value to be associated with the pset. The structure is designed to allow for expansion in the type of possible pset delimiters. An attrs attribute argument is included to allow for more detailed specification of how collective operations should be handled. Examples would include using a non-default radix, etc.

**Note:** This operation should not be invoked frequently by the application.

### 5.3.71.1 Synopsis

```c
dmapp_return_t dmapp_c_pset_export(
    IN dmapp_c_pset_handle_t *pset_handle);
```

### 5.3.71.2 Parameters

- **pset_handle**
  
  Pointer to an opaque dmapp_c_pset_handle_t structure, returned from previous call to dmapp_c_pset_create() to be used for subsequent DMAPP collective calls.

### 5.3.71.3 Return Codes

- **DMAPP_RC_SUCCESS**
  
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  
  One or more input parameters is invalid.

- **DMAPP_RC_NOT_FOUND**
  
  One or more of the processes associated with the pset handle has not invoked dmapp_c_pset_create.

### 5.3.72 dmapp_c_barrier_join

Initiate a barrier join operation on a pset. A successful return from this operation means the barrier operation is proceeding, not that it is complete.

#### 5.3.72.1 Synopsis

```c
dmapp_return_t dmapp_c_barrier_join(
    IN dmapp_c_pset_handle_t *pset_handle);
```

#### 5.3.72.2 Parameters

- **pset_handle**
  
  Structure from a previously exported pset.
5.3.72.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully. A successful return from this operation means the barrier operation is proceeding, not that it is complete.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_BUSY

The specified handle is currently in use for another collective operation.

5.3.73 dmapp_c_pset_cancel_op

Cancel an outstanding collective operation on a pset. This function has no effect if there is no outstanding collective operation. After a pset is canceled, it cannot be used for any further collective operations. The only allowed operation on a cancelled pset is dmapp_c_pset_destroy().

5.3.73.1 Synopsis

dmapp_return_t dmapp_c_pset_cancel_op(
    IN dmapp_c_pset_handle_t pset_handle);

5.3.73.2 Parameters

pset_handle

Structure from a previously created pset.

5.3.73.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

Input parameter is invalid.

5.3.74 dmapp_c_pset_destroy

Destroy a previously created pset, freeing DMAPP internal resources associated with the pset. This is a local operation, not an explicitly synchronizing function.
5.3.74.1 Synopsis

dmapp_return_t dmapp_c_pset_destroy(
    IN dmapp_c_pset_handle_t pset_handle);

5.3.74.2 Parameters

pset_handle

Structure from a previously created pset.

5.3.74.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

Input parameter is invalid.

DMAPP_RC_BUSY

The pset is still involved in a collective operation, in which case
an application should invoke this function at a later time to ensure
resources are released.

5.3.75 dmapp_c_greduce_start

Initiate a global reduction operation over the previously exported pset. A successful
return from this operation only means the operation is proceeding, not that it is
complete.

5.3.75.1 Synopsis

dmapp_return_t dmapp_c_greduce_start(
    IN dmapp_c_pset_handle_t *pset_handle,
    IN void *in,
    OUT void *out,
    IN uint32_t nelems,
    IN dmapp_c_type_t type,
    IN dmapp_c_op_t op);
5.3.75.2 Parameters

- **pset_handle**
  Pointer to an opaque dmapp_c_pset_handle_t structure from a previously exported pset.

- **in**
  Pointer to buffer containing this PE’s contribution to the global reduction over the pset.

- **out**
  Pointer to buffer where the result of the global reduction operation is to be returned.

- **nelems**
  Number of elements in the global operation. The number of elements specified cannot exceed the value returned by dmapp_c_greduce_nelems_max for the given data type and operation type.

- **type**
  Data type of the elements in the global operation. See dmapp_c_type on page 144.

- **op**
  Type of global reduction operation. See dmapp_c_op on page 144.

5.3.75.3 Return Codes

- **DMAPP_RC_SUCCESS**
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  One or more input parameters is invalid.

- **DMAPP_RC_BUSY**
  The specified handle is currently in use for another collective operation.

5.3.76 dmapp_c_greduce_nelems_max

Return the maximum number of elements of a given data type which can be used for a single global reduction operation. This function may be used before a dmapp initialization call is made.

5.3.76.1 Synopsis

```c
dmapp_return_t dmapp_c_greduce_nelems_max(
    IN dmapp_c_type_t type,
    OUT uint32_t *nelems_max);
```
5.3.76.2 Parameters

*type*  
Data type of the elements in the global operation. See [dmapp_c_type on page 144](#).

*nelems_max*  
Maximum number of elements of the supplied data type which can be used for a single global reduction operation.

5.3.76.3 Return Codes

- **DMAPP_RC_SUCCESS**
  
  The operation completed successfully.

- **DMAPP_RC_INVALID_PARAM**
  
  One or more input parameters is invalid.

5.3.77 dmapp_c_pset_test

Test for completion of a collective operation for a given *pset* in a non-blocking manner.

5.3.77.1 Synopsis

```c
dmapp_c_pset_test(IN dmapp_c_pset_handle_t pset_handle)
```

5.3.77.2 Parameters

*pset_handle*

*pset* for which a collective operation has been initiated.

5.3.77.3 Return Codes

- **DMAPP_RC_SUCCESS**
  
  The collective operation completed locally. Local completion of a previously initiated collective operation does not mean all ranks have completed the collective operation, but that all ranks have initiated the operation.

- **DMAPP_RC_INVALID_PARAM**
  
  Input parameter is invalid.

- **DMAPP_RC_NOT_DONE**
  
  Collective operation has not completed locally.

- **DMAPP_RC_TRANSACTION_ERROR**
  
  An unrecoverable network error was encountered.
5.3.78 dmapp_c_pset_wait

Wait for completion of a collective operation for a given pset. This function is the blocking equivalent of dmapp_c_pset_test.

5.3.78.1 Synopsis

dmapp_c_pset_wait(IN dmapp_c_pset_handle_t pset_handle);

5.3.78.2 Parameters

pset_handle

pset for which a collective operation has been initiated.

5.3.78.3 Return Codes

DMAPP_RC_SUCCESS

The collective operation completed successfully.

DMAPP_RC_INVALID_PARAM

Input parameter is invalid.

DMAPP_RC_NOT_DONE

The collective operation has not completed.

DMAPP_RC_TRANSACTION_ERROR

An unrecoverable network error was encountered.

5.3.79 dmapp_sheap_malloc

The dmapp_sheap_malloc function allocates size bytes of memory from the symmetric heap. The space returned is left uninitialized. It cannot be assumed that the memory returned is zeroed out. There are no address equality guarantees across ranks.

5.3.79.1 Synopsis

void *dmapp_sheap_malloc(
    IN size_t size);

5.3.79.2 Parameters

size

The size, in bytes, of memory to allocate from the symmetric heap.
5.3.80 dmapp_sheap_realloc

The dmapp_sheap_realloc function changes the size of the block to which `ptr` points to the size, in bytes, specified by `size`. The contents of the block are unchanged up to the lesser of the new and old sizes. If the new size is larger, the value of the newly allocated portion of the block is indeterminate. If `ptr` is a null pointer, dmapp_sheap_realloc behaves like dmapp_sheap_malloc for the specified size. If `size` is 0 and `ptr` is not a null pointer, the block to which it points is freed. Otherwise, if `ptr` does not match a pointer earlier returned by a symmetric heap function, or if the space has already been deallocated, dmapp_sheap_realloc returns a null pointer. If the space cannot be allocated, the block to which `ptr` points is unchanged.

5.3.80.1 Synopsis

```c
void *dmapp_sheap_realloc(
    IN void  *ptr,
    IN size_t size);
```

5.3.80.2 Parameters

- `ptr` Pointer to the block.
- `size` The size, in bytes, of which to change the block.

5.3.81 dmapp_sheap_free

The dmapp_sheap_free function frees a block of memory previously allocated by dmapp_sheap_malloc or dmapp_sheap_realloc.

5.3.81.1 Synopsis

```c
void dmapp_sheap_free(
    IN void  *ptr);
```

5.3.81.2 Parameters

- `ptr` Pointer to the block of memory previously allocated with dmapp_sheap_malloc or dmapp_sheap_realloc.

5.3.82 dmapp_mem_register

The dmapp_mem_register function dynamically registers a memory region, other than statically linked data segment or the symmetric heap, with the NIC.

The memory region is described by starting address `addr` and `length`. This memory could have been allocated from the private heap or using `mmap`. Memory registered by a call to dmapp_mem_register becomes remotely accessible and is assumed to be non-symmetric.
The function updates the content of the segment descriptor to reflect the actual starting address and length of the region which was registered. These values can differ from the input values due to rounding, for instance. Dynamically registered memory can only be remotely accessed by point-to-point RMA functions, not by PE-strided RMA functions.

5.3.82.1 Synopsis

```c
dmapp_return_t dmapp_mem_register(
    IN void *addr,
    IN uint64_t length,
    INOUT dmapp_seg_desc_t *seg_desc);
```

5.3.82.2 Parameters

- **addr**: Points to starting address of the memory region to be registered. Must be non-NULL.
- **length**: Length in bytes of the memory region in bytes. Must be > 0.
- **seg_desc**: On input, points to segment descriptor and must be non-NULL. Function updates it to the actual starting address and length of the registered region.

5.3.82.3 Return Codes

- **DMAPP_RC_SUCCESS**: The operation completed successfully.
- **DMAPP_RC_RESOURCE_ERROR**: Unsuccessful memory registration or invalid memory handle.
- **DMAPP_RC_INVALID_PARAM**: One or more input parameters is invalid.

5.3.83 dmapp_mem_unregister

The `dmapp_mem_unregister` function unregisters a memory region, other than statically linked data segment or the symmetric heap, on the fly, from the NIC. The memory region must previously have been registered by a call to `dmapp_mem_register`.

5.3.83.1 Synopsis

```c
dmapp_return_t dmapp_mem_unregister(
    IN dmapp_seg_desc_t *seg_desc);
```
5.3.83.2 Parameters

\textit{seg\_desc} Points to segment descriptor to deregister, which must have been registered with a call to \texttt{dmapp\_mem\_register}.

5.3.83.3 Return Codes

\begin{itemize}
\item \texttt{DMAPP\_RC\_SUCCESS} \\
The operation completed successfully.
\item \texttt{DMAPP\_RC\_INVALID\_PARAM} \\
Input parameter is invalid.
\end{itemize}

5.3.84 \texttt{dmapp\_segdesc\_compare}

The \texttt{dmapp\_segdesc\_compare} function compares two segment descriptors. If they describe the same memory region, flag is set to 1. If they describe different memory regions, flag is set to 0.

5.3.84.1 Synopsis

\begin{verbatim}
dmapp_return_t dmapp_segdesc_compare(
    IN  dmapp_seg_desc_t *seg_desc1,
    OUT dmapp_seg_desc_t *seg_desc2,
    INOUT int *flag);
\end{verbatim}

5.3.84.2 Parameters

\begin{itemize}
\item \textit{seg\_desc1} Pointer to segment descriptor 1.
\item \textit{seg\_desc2} Pointer to segment descriptor 2.
\item \textit{flag} set to 1 if both segment descriptors describe the same memory region, otherwise set to 0.
\end{itemize}

5.3.84.3 Return Codes

\begin{itemize}
\item \texttt{DMAPP\_RC\_SUCCESS} \\
The operation completed successfully.
\item \texttt{DMAPP\_RC\_INVALID\_PARAM} \\
One or more input parameters is invalid.
5.3.85 dmapp_register_process_cb

Register a process callback function. This function is used when there are progress requirements that require the use of DMAPP even when using blocking DMAPP calls, e.g. dmapp_get. A progress callback function should return 0 upon success, -1 upon failure.

5.3.85.1 Synopsis

dmapp_return_t dmapp_register_process_cb(
    IN int *progress_cb,
    IN void *data);

5.3.85.2 Parameters

progress_cb

Pointer to callback function to be registered.

data

Pointer to optional data supplied as an argument to the callback function. Can be null.

5.3.85.3 Return Codes

DMAPP_RC_SUCCESS

The operation completed successfully.

DMAPP_RC_INVALID_PARAM

One or more input parameters is invalid.

DMAPP_RC_NO_SPACE

Too many callback functions already registered.

5.3.86 dmapp_deregister_process_cb

Deregister a progress callback function previously registered using dmapp_register_process_cb.

5.3.86.1 Synopsis

dmapp_return_t dmapp_deregister_process_cb(
    IN int *progress_cb);

5.3.86.2 Parameters

progress_cb

Pointer to callback function to be deregistered.
5.3.86.3 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_INVALID_PARAM
Input parameter is invalid.

DMAPP_RC_NO_SPACE
Supplied callback function not found in callback list.

5.3.87 dmapp_checkpoint

The dmapp_checkpoint function destroys GNI resources and DMAPP internal resources that cannot be checkpointed. This function should not be called outside of the checkpoint-restart context. In a threaded environment, this function should only be called once per process.

5.3.87.1 Synopsis

dmapp_return_t dmapp_checkpoint(void);

5.3.87.2 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_NOT_DONE
Upper-level software has not quiesced the network.

5.3.88 dmapp_restart

This is a checkpoint-restart specific function and should not be called outside the context of that feature. Reinitialize GNI and DMAPP internal resources which could not be checkpointed. In a threaded environment, this function should only be called once per process.

5.3.88.1 Synopsis

dmapp_return_t dmapp_restart(
    IN uint32_t restart_modes);

5.3.88.2 Parameters

restart_modes
5.3.88.3 Return Codes

DMAPP_RC_SUCCESS
The operation completed successfully.

DMAPP_RC_NOT_DONE
Upper-level software has not restarted the network.

5.4 Environment Variables Which Affect DMAPP

5.4.1 XT_SYMmetric_HEAP_SIZE
Controls the size (in bytes) of the symmetric heap. One Mbyte is allocated for internal use only.
Default: 0 bytes for the user

5.4.2 DMAPP_ABORT_ON_ERROR
Allows a user to force a core dump upon error. This is supported during initialization and memory handling operations.
Default: not set

5.4.3 DMAPP_PUT_NBI_CHAIN_OFF
By default, message rate optimization for non-blocking implicit PUTs and non-fetching AMOs is on. Defining this environment variable turns it off.
A.1 dmapp_put.c

#include <stdio.h>
#include <unistd.h>
#include "pmi.h"
#include "dmapp.h"

#define MAX_NELEMS (128L*1024L)

/* If necessary, run the job with fewer than the maximum number of cores
 * per node so that enough memory is available for each PE. */

int main(int argc, char **argv)
{
    int pe = -1;
    int npes = -1;
    int target_pe;
    int fail_count = 0;
    long nelems = MAX_NELEMS;
    long *source;
    long *target;
    long i;
    dmapp_return_t status;
    dmapp_rma_attrs_t actual_args = {0}, rma_args = {0};
    dmapp_jobinfo_t job;
    dmapp_seg_desc_t *seg = NULL;

    /* Set the RMA parameters. */

    rma_args.put_relaxed_ordering = DMAPP_ROUTING_ADAPTIVE;
    rma_args.max_outstanding_nb = DMAPP_DEF_OUTSTANDING_NB;
    rma_args.offload_threshold = DMAPP_OFFLOAD_THRESHOLD;
    rma_args.max_concurrency = 1;

    /* Initialize DMAPP. */

    status = dmapp_init(&rma_args, &actual_args);
    if (status != DMAPP_RC_SUCCESS) {
        fprintf(stderr, " dmapp_init FAILED: %d\n", status);
        exit(1);
    }

    /* Allocate and initialize the source and target arrays. */

    source = (long *)dmapp_sheap_malloc(nelems*sizeof(long));
    target = (long *)dmapp_sheap_malloc(nelems*sizeof(long));
    if ((source == NULL) || (target == NULL)) {
        fprintf(stderr, " sheap_malloc’d failed src 0x%lx targ 0x%lx\n",
                 (long)source, (long)target);
        exit(1);
    }
}

exit(1);
}
for (i=0; i<nelems; i++) {
    source[i] = i;
    target[i] = -9L;
}
/* Wait for all PEs to complete array initialization. */
PMI_Barrier();
/* Get job related information. */
status = dmapp_get_jobinfo(&job);
if (status != DMAPP_RC_SUCCESS) {
    fprintf(stderr," dmapp_get_jobinfo FAILED: %d\n", status);
    exit(1);
}
pe = job.pe;
npes = job.npes;
seg = & (job.sheap_seg);
/* Send my data to my target PE. */
target_pe = npes - pe -1;
status = dmapp_put(target, seg, target_pe, source, nelems, DMAPP_QW);
if (status != DMAPP_RC_SUCCESS) {
    fprintf(stderr," dmapp_put FAILED: %d\n", status);
    exit(1);
}
/* Wait for all PEs to complete their PUT. */
PMI_Barrier();
/* Check the results. */
for (i=0; i<nelems; i++) {
    if ((target[i] != i) && (fail_count<10)) {
        fprintf(stderr," PE %d: target[%ld] is %ld, should be %ld\n",
                      pe, i, target[i], (long)i);
        fail_count++;
    }
}
if (fail_count == 0) {
    fprintf(stderr," dmapp_put PASSED for PE %04d\n", pe);
} else {
    fprintf(stderr," dmapp_put FAILED for PE %04d: 
" "%d or more wrong values\n", pe, fail_count);
}
/* Finalize. */
status = dmapp_finalize();
return(0);
}